WILLIAM STALLINGS

ORGANIZACJA I ARCHITEKTURA SYSTEMU KOMPUTEROWEGO

Projektowanie systemu a jego wydajność

Wydawnictwa Naukowo-Techniczne

z angielskiego
przełożył
Jacek B. Szporko

55 lat WNT

WILLIAM STALLINGS

ORGANIZACJA I ARCHITEKTURA SYSTEMU KOMPUTEROWEGO

Projektowanie systemu a jego wydajność

Wydanie trzecie zmienione i rozszerzone

Wydawnictwa Naukowo-Techniczne • Warszawa

Dane o oryginale
William Stallings
COMPUTER ORGANIZATION AND ARCHITECTURE
DESIGNING FOR PERFORMANCE
Sixth Edition

Authorized translation from the English language edition, entitled COMPUTER ORGANIZATION AND ARCHITECTURE: DESIGNING FOR PERFORMANCE, Sixth Edition by STALLINGS, WILLIAM, published by Pearson Education, Inc., publishing as Prentice Hall

Polish language edition published by WYDAWNICTWA NAUKOWO-TECHNICZNE

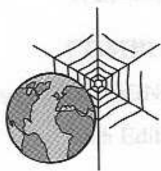
Redaktor pierwszego wydania *Izabela Ewa Mika*
Redaktor trzeciego wydania *Zuzanna Grzejszczak*
Okładkę i strony tytułowe projektował *Andrzej Pilich*
Redaktor techniczny *Marta Jeczeń-Bańkowska*
Korekta *Zespół*
Skład i łamanie *Marianna Zadrożna*

Podręcznik akademicki dotowany przez
Ministerstwo Edukacji Narodowej i Sportu

Wydawnictwa Naukowo-Techniczne
00-048 Warszawa, ul. Mazowiecka 2/4
tel. (0-22) 826 72 71, e-mail: wnt@pol.pl
www.wnt.com.pl

ISBN 83-204-2892-0

Jak zawsze, dla A.T.S.



Witryna WWW związana z szóstym angielskim wydaniem *Organizacji i architektury systemu komputerowego*

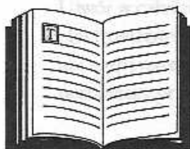
Witryna WWW pod adresem WilliamStallings.com/COA6e.html wspomaga wykładowców i studentów posługujących się tą książką. Obejmuje ona niżej wymienione elementy.



Materiały pomocnicze do wykładów

Materiały te obejmują:

- Kopie zawartych w książce rysunków w formacie PDF.
- Kopie zawartych w książce tabel w formacie PDF.
- Zbiór slajdów PowerPoint przeznaczonych dla wykładowców.
- Zbiór notatek szkoleniowych PDF, które mogą być wręczane studentom lub mogą służyć jako przezrocza.
- Witrynę z materiałami dla studentów informatyki (*Computer Science Student Resource Site*). Zawiera ona łącza i dokumenty, które mogą okazać się przydatne studentom kształcącym się w dziedzinie informatyki. Znajduje się w niej przegląd podstawowych problemów matematyki; porady dotyczące wyszukiwania informacji, pisania i rozwiązywania prac domowych; łącza do zasobów informacyjnych związanych z informatyką, takich jak składnice raportów i bibliografie, a także inne przydatne łącza.
- Errata do tej książki, aktualizowana najrzadziej co miesiąc.



Wykłady na temat organizacji i architektury systemu komputerowego

Witryna WWW COA6e zawiera łącza do witryn związanych z wykładami opartymi na tej książce. W witrynach tych znajdują się przydatne informacje dotyczące terminów i tematyki wykładów, a także wiele użytecznych materiałów pomocniczych.



Przydatne witryny WWW

W witrynie COA6e znajdują się łącza do innych witryn o szerokim zakresie tematycznym, które ułatwią studentom głębsze zapoznawanie się z aktualnymi zagadnieniami.



Internetowa lista adresowa

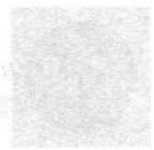
Prowadzona jest internetowa lista adresowa, dzięki której wykładowcy posługujący się tą książką mogą wymieniać informacje, sugestie i pytania między sobą i z udziałem autora. Informacje na temat sposobu przyłączenia się do tej listy znajdują się w witrynie WWW książki.



Narzędzia symulacyjne dla zadań związanych z organizacją i architekturą systemu komputerowego

W witrynie książki znajdują się łącza do witryn WWW związanych z narzędziami SimpleScalar i SMPCache. Są to dwa pakiety oprogramowania wspomagającego implementację przedsięwzięć. Z każdej witryny można pobierać oprogramowanie i informacje pomocnicze. Więcej informacji na ten temat znajduje się w Dodatku C.





Obejmuje one niżej wymienione elementy.





Materiały pomocnicze do wykładów

- Zbiór slajdów PowerPoint przeznaczonych dla wykładowców.
- Zbiór notatek szkoleniowych PDF, które mogą być rozdawane studentom lub mogą służyć jako prezentacja.





Wykłady na temat organizacji i architektury systemu komputerowego

Spis treści

Przedmowa 15

CZĘŚĆ PIERWSZA: PRZEGLĄD 21

Rozdział 1. Wprowadzenie 23

1.1. Organizacja i architektura 24
1.2. Struktura i działanie 25
1.3. Dlaczego warto studiować organizację i architekturę komputerów? 32
1.4. Zawartość książki 33
1.5. Zasoby Internetu i sieci WWW 34

Rozdział 2. Ewolucja i wydajność komputerów 37

2.1. Krótka historia komputerów 38
2.2. Projektowanie zorientowane na wydajność 60
2.3. Ewolucja procesorów Pentium i PowerPC 65
2.4. Polecana literatura i witryny WWW 68
2.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 69

CZĘŚĆ DRUGA: SYSTEM KOMPUTEROWY 71

Rozdział 3. Ogólny obraz działania komputera i jego połączeń wewnętrznych 73

3.1. Zespoły komputera 75
3.2. Działanie komputera 78
3.3. Struktura połączeń wewnętrznych 92
3.4. Połączenia magistralowe 94
3.5. Magistrala PCI 105
3.6. Polecana literatura i witryny WWW 115
3.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 116

Dodatek 3A: Wykresy przebiegów czasowych 118

Rozdział 4. Pamięć podręczna 121

4.1. Przegląd systemów pamięci komputerowych 123
4.2. Zasady działania pamięci podręcznej 130
4.3. Elementy projektowania pamięci podręcznych 133
4.4. Organizacja pamięci podręcznej w Pentium 4 i PowerPC 147
4.5. Polecana literatura 152
4.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 152
Dodatek 4A: Charakterystyka wydajności pamięci dwupoziomowych 156

Rozdział 5. Pamięć wewnętrzna 165

5.1. Półprzewodnikowa pamięć główna 166
5.2. Korekcja błędów 177
5.3. Nowe rozwiązania organizacji DRAM 182
5.4. Polecana literatura i witryny WWW 188
5.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 189

Rozdział 6. Pamięć zewnętrzna 191

6.1. Dysk magnetyczny 192
6.2. RAID 203
6.3. Pamięć optyczna 214
6.4. Taśma magnetyczna 219
6.5. Polecana literatura i witryny WWW 221
6.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 222

Rozdział 7. Wejście-wyjście 225

7.1. Urządzenia zewnętrzne 227
7.2. Moduły wejścia-wyjścia 232
7.3. Programowane wejście-wyjście 236
7.4. Wejście-wyjście sterowane przerwaniami 240
7.5. Bezpośredni dostęp do pamięci 249
7.6. Kanały i procesory wejścia-wyjścia 253
7.7. Interfejs zewnętrzny: FireWire i InfiniBand 255
7.8. Polecana literatura i witryny WWW 267
7.9. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 268

Rozdział 8. Wspieranie systemu operacyjnego 271

8.1. Przegląd systemów operacyjnych 273
8.2. Szeregowanie 285
8.3. Zarządzanie pamięcią 292
8.4. Zarządzanie pamięcią w Pentium II i PowerPC 304
8.5. Polecana literatura i witryny WWW 314
8.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 314

CZĘŚĆ TRZECIA: JEDNOSTKA CENTRALNA 319

Rozdział 9. Arytmetyka komputera 321

9.1. Jednostka arytmetyczno-logiczna 322
9.2. Reprezentacja liczb całkowitych 323
9.3. Arytmetyka liczb całkowitych 329
9.4. Reprezentacja zmiennopozycyjna 346
9.5. Arytmetyka zmiennopozycyjna 352
9.6. Polecana literatura i witryny WWW 362
9.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 363

Rozdział 10. Listy rozkazów: własności i funkcje 369

10.1. Właściwości rozkazów maszynowych 371
10.2. Rodzaje argumentów 377
10.3. Rodzaje danych Pentium i PowerPC 380
10.4. Rodzaje operacji 383
10.5. Rodzaje operacji Pentium i PowerPC 398
10.6. Język asemblerowy 408
10.7. Polecana literatura 410
10.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 411
Dodatek 10A: Stosy 416
Dodatek 10B: Struktury „cienko-", „grubo-" i „dwukońcowe" 421

Rozdział 11. Listy rozkazów: tryby adresowania i formaty rozkazów 427

11.1. Adresowanie 428
11.2. Tryby adresowania Pentium i PowerPC 436
11.3. Formaty rozkazów 442
11.4. Formaty rozkazów Pentium i PowerPC 452
11.5. Polecana literatura 457
11.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 458

Rozdział 12. Struktura i działanie procesora 461

12.1. Organizacja procesora 462
12.2. Organizacja rejestrów 464
12.3. Cykl rozkazu 470
12.4. Potokowe przetwarzanie rozkazów 474
12.5. Procesor Pentium 491
12.6. Procesor PowerPC 500
12.7. Polecana literatura 507
12.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 508

Rozdział 13. Komputery o zredukowanej liście rozkazów 511

13.1. Właściwości wykonywania rozkazów 513
13.2. Użycie dużej tablicy rejestrów 518

13.3. Optymalizacja rejestrów za pomocą kompilatora 524
13.4. Architektura o zredukowanej liście rozkazów 526
13.5. Przetwarzanie potokowe w architekturze RISC 533
13.6. MIPS R4000 537
13.7. SPARC 552 546
13.8. Porównanie architektur RISC i CISC 552
13.9. Polecana literatura 554
13.10. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 554

Rozdział 14. Paralelizm na poziomie rozkazu i procesory superskalarne 559

14.1. Przegląd 561
14.2. Problemy projektowania 566
14.3. Pentium 4 576
14.4. PowerPC 585
14.5. Polecana literatura 593
14.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 595

Rozdział 15. Architektura IA-64 599

15.1. Uzasadnienie 601
15.2. Organizacja ogólna 602
15.3. Predykacja, spekulacja i potokowanie programowe 604
15.4. Architektura listy rozkazów IA-64 624
15.5. Organizacja Itanium 629
15.6. Polecana literatura i witryny WWW 630
15.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 632

CZĘŚĆ CZWARTA: JEDNOSTKA STERUJĄCA 635

Rozdział 16. Działanie jednostki sterującej 637

16.1. Mikrooperacje 639
16.2. Sterowanie procesorem 645
16.3. Implementacja układowa 658
16.4. Polecana literatura 660
16.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 661

Rozdział 17. Sterowanie mikroprogramowe 663

17.1. Koncepcje podstawowe 664
17.2. Szeregowanie mikrorozkazów 673
17.3. Wykonywanie mikrorozkazów 680
17.4. Moduł TI 8800 692
17.5. Zastosowania mikroprogramowania 703
17.6. Polecana literatura 704
17.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 705

CZĘŚĆ PIĄTA: ORGANIZACJA RÓWNOLEGŁA 707

Rozdział 18. Przetwarzanie równoległe 709

18.1. Organizacje wieloprocesorowe 711
18.2. Wieloprocesory symetryczne 714
18.3. Spójność pamięci podręcznej i protokół MESI 724
18.4. Klastry 732
18.5. Niejednorodny dostęp do pamięci 739
18.6. Obliczenia wektorowe 744
18.7. Polecana literatura 758
18.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 759

Dodatek A. Logika cyfrowa 765

A.1. Algebra Boole'a 766
A.2. Bramki 768
A.3. Układy kombinacyjne 771
A.4. Układy sekwencyjne 793
A.5. Problemy do rozwiązania 804

Dodatek B. Systemy liczbowe 807

B.1. System dziesiętny 808
B.2. System binarny 808
B.3. Konwersja między liczbami dziesiętnymi a binarnymi 809
B.4. Notacja szesnastkowa 812
B.5. Problemy do rozwiązania 813

Dodatek C. Zadania ułatwiające nauczanie organizacji i architektury komputera 815

C.1. Wyszukiwanie informacji 816
C.2. Zadania symulacyjne 817
C.3. Zadania typu lektura-sprawozdanie 818

Słownik 819
Literatura 831
Skorowidz 843

13.3. Optymalizacja rejestrów ...
13.4. Architektura o zredukowanej liście rozkazów
13.5. Przetwarzanie potokowe w architekturze RISC
13.6. MIPS R4000 537
13.7. SPARC 546
13.8. Porównanie architektur RISC i CISC
13.9. Polecana literatura 553
13.10. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Rozdział 14. Paralelizm na poziomie rozkazu i procesory superskalarne

14.1. Przegląd 561
14.2. Problemy projektowania 568
14.3. Pentium 4 576
14.4. PowerPC 585
14.5. Polecana literatura 593
14.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Rozdział 15. Architektura IA-64 599

15.1. Uzasadnienie 601
15.2. Organizacja ogólna 602
15.3. Predykacja, spekulacja i potokowanie programowe
15.4. Architektura listy rozkazów IA-64
15.5. Organizacja Itanium 629
15.6. Polecana literatura i witryny WWW
15.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

CZĘŚĆ CZWARTA: JEDNOSTKA STERUJĄCA 635

Rozdział 16. Działanie jednostki sterującej 637

16.1. Mikrooperacje 639
16.2. Sterowanie procesorem 645
16.3. Implementacja układowa 656
16.4. Polecana literatura 660
16.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Rozdział 17. Sterowanie mikroprogramowe 663

17.1. Koncepcje podstawowe 664
17.2. Szeregowanie mikrorozkazów 675
17.3. Wykonywanie mikrorozkazów 680
17.4. Moduł TI 8800 692
17.5. Zastosowania mikroprogramowania 703
17.6. Polecana literatura 704
17.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania 705

Przedmowa

Cele

Oto książka o strukturze i działaniu komputerów. Pisząc ją, chciałem możliwie jasno i wyczerpująco przedstawić naturę i własności nowoczesnych systemów komputerowych.

Z wielu powodów zadanie to jest wyzwaniem. Po pierwsze, istnieje ogromna różnorodność wyrobów, które mogą nosić miano komputera – od pojedynczego mikroprocesora za kilka dolarów do superkomputerów kosztujących dziesiątki milionów dolarów. Różnorodność dotyczy nie tylko kosztów, ale również rozmiarów, wydajności i obszaru zastosowań. Po drugie, szybkie zmiany, które zawsze charakteryzowały technikę komputerową, zachodzą nadal i wcale nie słabną. Zmiany te obejmują wszystkie aspekty techniki komputerowej – od podstawowej technologii układów scalonych, służących do budowy zespołów komputera, do coraz większego wykorzystania koncepcji organizacji równoległej przy łączeniu tych zespołów.

Pomimo tej różnorodności i tempa zmian w dziedzinie komputerów pewne podstawowe rozwiązania pozostają nadal aktualne. Z pewnością jednak zastosowanie ich zależy od bieżącego stanu techniki oraz od założonej przez projektanta relacji między ceną a wydajnością. Intencją moją było dogłębne omówienie podstaw organizacji i architektury komputerów oraz ich powiązanie ze współczesnymi zagadnieniami projektowania.

Podtytuł odzwierciedla myśl przewodnią tej książki i przyjęte w niej podejście. Osiąganie dużej wydajności było zawsze ważne przy projektowaniu systemów komputerowych. Nigdy jednak wymaganie to nie było tak ważne i jednocześnie tak trudne do spełnienia jak dziś. Wartości wszystkich podstawowych parametrów związanych z wydajnością systemów komputerowych, w tym szybkości procesora, szybkości pamięci, jej pojemności oraz szybkości przepływu danych, stale rosną. Ponadto ich wzrost następuje w różnym tempie. Utrudnia to zaprojektowanie zrównoważonego systemu, który maksymalnie wykorzystuje wydajność wszystkich elementów. W tej sytuacji projektowanie komputera w coraz większym stopniu polega na zmianie struktury lub działania w jednym obszarze w celu skompensowania niezrówno-

ważenia wydajności w innym. Tego rodzaju postępowanie będzie widoczne przy licznych zagadnieniach projektowania przedstawionych w tej książce.

System komputerowy, tak jak każdy system, składa się z współzależnych zespołów elementów. Najlepiej można go scharakteryzować przez określenie jego struktury (czyli sposobu powiązania zespołów) i określenie jego funkcjonowania (czyli działania poszczególnych zespołów). Ponadto należy uwzględnić to, że struktura komputera jest hierarchiczna. Każdy główny zespół można analizować dalej, rozkładając go na główne podzespoły i opisując ich strukturę oraz działanie. Uznałem, że bardziej przejrzyste i łatwiejsze do zrozumienia będzie przedstawienie tej hierarchicznej organizacji w sposób „od ogółu do szczegółu".

- **System komputerowy.** Główne zespoły to procesor, pamięć i urządzenia wejścia-wyjścia.
- **Procesor.** Główne zespoły to jednostka sterująca, rejestry, jednostka arytmetyczno-logiczna i jednostka wykonująca rozkazy.
- **Jednostka sterująca.** Główne zespoły to pamięć sterowania, zespół szeregowania mikrorozkazów i rejestry.

Bardzo zależało mi na jak najprostszym przedstawieniu nowego materiału. Sądzę, że dzięki przyjętemu przeze mnie podejściu „od ogółu do szczegółu" czytelnik się w tym wszystkim nie pogubi.

System jest rozpatrywany zarówno pod względem architektury (to znaczy atrybutów systemu widzialnych dla programującego w języku maszynowym), jak i organizacji (to znaczy jednostek operacyjnych i ich połączeń tworzących architekturę).

Przykładowe systemy

W celu wyjaśnienia przedstawionych koncepcji, a także potwierdzenia ich słuszności wykorzystałem przykłady pochodzące z wielu różnych maszyn, głównie zaś z dwóch rodzin komputerów: Pentium 4 firmy Intel oraz PowerPC firm IBM i Motorola. Te właśnie dwa systemy komputerowe wyznaczają większość bieżących tendencji w projektowaniu. Pentium 4 to w zasadzie komputery o złożonej liście rozkazów (CISC) z niektórymi właściwościami komputerów o zredukowanej liście rozkazów (RISC), podczas gdy PowerPC to w zasadzie komputery o zredukowanej liście rozkazów. W obu systemach wykorzystano zasady projektowania superskalarnego i oba umożliwiają stosowanie konfiguracji wieloprocesorowej.

Układ książki

Książka jest podzielona na pięć części.

Część pierwsza – Przegląd. Zawiera przegląd zagadnień omawianych w książce.

Część druga – System komputerowy. System komputerowy składa się z procesora, pamięci i urządzeń wejścia-wyjścia oraz z połączeń między tymi głównymi zespoła-

mi. Omówiłem tu kolejno każdy z tych zespołów z wyjątkiem procesora. Jest on na tyle złożony, że poświęciłem mu część trzecią.

Część trzecia – Jednostka centralna. Jednostka centralna składa się z jednostki sterującej, rejestrów, jednostki arytmetyczno-logicznej, jednostki wykonywania rozkazów oraz połączeń między tymi zespołami. Opisałem tu zagadnienia dotyczące architektury, takie jak projektowanie list rozkazów i rodzaje danych, a także zagadnienia organizacyjne, takie jak przetwarzanie potokowe.

Część czwarta – Jednostka sterująca. Jednostka sterująca jest tym elementem procesora, który uruchamia różne zespoły. Ta część książki dotyczy funkcjonowania jednostki sterującej i jej realizacji za pomocą mikroprogramowania.

Część piąta – Organizacja równoległa. Tę część poświęciłem na omówienie zagadnień dotyczących organizacji wieloprocesorowej i organizacji wykorzystującej przetwarzanie wektorowe.

Książka zawiera również obszerny słownik, zestawienie często używanych akronimów i bibliografię. Do każdego rozdziału dodałem problemy przeznaczone do rozwiązania w domu, pytania kontrolne, wykaz podstawowych pojęć, sugerowane lektury dodatkowe i zalecane witryny WWW.

Dokładniejsze streszczenie poszczególnych części książki – rozdział po rozdziale – znajduje się na początku każdej części.

Adresaci książki

Książka jest przeznaczona zarówno dla środowiska akademickiego, jak i dla osób zajmujących się informatyką profesjonalnie. Jako podręcznik może być podstawą jedno- lub dwusemestralnego wykładu na wydziałach informatyki, techniki komputerowej i elektrotechniki. Obejmuje wszystkie tematy przewidziane w wykładzie *CS220 Computer Architecture*, wchodzącym w skład *IDEE/ACM Computer Curricula 2001* [JTF01].

Specjalistom z tej dziedziny książka ta może służyć jako podstawowe kompendium wiedzy i jako pomoc do samodzielnej nauki.

Usługi internetowe dla wykładowców i studentów

Istnieje związana z tą książką witryna WWW, wspomagająca wykładowców i studentów. Zawiera łącza prowadzące do innych witryn, kopie zawartych w tej książce rysunków i tablic w formacie PDF (Adobe Acrobat) oraz informacje o sposobie subskrybowania materiałów z internetowej listy adresowej. Witryna ta znajduje się pod adresem WilliamStallings.com/COA6e.html; więcej informacji na jej temat znajduje się w poprzedzającym tę przedmowę materiale pt. „Witryna WWW związana z angielskim szóstym wydaniem książki *Organizacja i architektura systemu komputerowego*". Została utworzona internetowa lista adresowa, dzięki której wy-

kładowcy posługujący się tą książką mogą wymieniać informacje, sugestie i pytania między sobą, a także z autorem. Gdy tylko zostaną wykryte pomyłki drukarskie i inne błędy, w witrynie WilliamStallings.com. będzie udostępniona errata do tej książki. Ponadto pod adresem WilliamStallings.com/StudentSupport.html znajdują się materiały przeznaczone dla studentów informatyki, a w tym przydatne dokumenty, informacje i łącza.

Zadania przydatne w nauczaniu organizacji i architektury systemu komputerowego

Zdaniem wielu wykładowców ważną częścią zajęć poświęconych organizacji i architekturze systemu komputerowego są zadania (przedsięwzięcia), dzięki którym student może praktycznie zgłębić zawarte w podręczniku koncepcje. Książka ta w wyjątkowy zupełnie sposób umożliwia uzupełnianie wykładów tego rodzaju zadaniami. Podręcznik wykładowcy zawiera nie tylko wskazówki dotyczące sposobu wyznaczania i formułowania zadań, ale też zestaw zadań pokrywający szeroki zakres tematyczny.

- **Zadania badawcze.** Podręcznik zawiera wiele zadań, w ramach których studenci mają znaleźć w sieci WWW lub w literaturze informacje na określony temat i napisać sprawozdanie.
- **Zadania symulacyjne.** W podręczniku jest uwzględnione użycie dwóch pakietów symulacyjnych. SimpleScalar służy do zgłębiania problemów projektowania organizacji i architektury komputerów. SMPCache jest potężnym narzędziem edukacyjnym, ułatwiającym rozważanie problemów projektowania pamięci podręcznych w odniesieniu do wieloprocesorów symetrycznych.
- **Zadania polegające na studiowaniu zalecanych lektur.** Podręcznik zawiera wykaz prac (co najmniej jeden na każdy rozdział), które student powinien przeczytać, a następnie napisać krótkie sprawozdanie.

Szczegółowe informacje na ten temat znajdują się w dodatku C.

Co nowego w angielskim szóstym wydaniu

W ciągu trzech lat, które upłynęły od piątego wydania tej książki, wprowadzono w rozpatrywanej dziedzinie liczne innowacje i ulepszenia. W nowym, szóstym wydaniu starałem się je uchwycić mając na uwadze obszerne i wyczerpujące przedstawienie całej dziedziny. Uwzględnienie tych wszystkich zmian zostało poprzedzone zebraniem opinii wielu wykładających ten przedmiot profesorów. Ponadto poszczególne rozdziały zostały zrecenzowane przez osoby zajmujące się informatyką zawodowo. W rezultacie w wielu miejscach narracja stała się bardziej przejrzysta i precyzyjna, a rysunki dokładniejsze. Dodałem też wiele zagadnień wynikających z praktyki.

Oprócz tych udoskonaleń, mających na celu poprawienie książki od strony dydaktycznej i uczynienie jej bardziej zrozumiałą, dokonałem wielu innych istotnych

zmian. Zachowałem niemal taką samą organizację rozdziałów, ale znaczną część materiału przeredagowałem i dodałem sporo nowego. Oto niektóre, najbardziej godne uwagi zmiany.

- **Architektura IA-64 (Itanium).** W tej nowej architekturze są uwzględnione tak ważne koncepcje, jak predykatywne wykonywanie i spekulatywne ładowanie rozkazów. Opis i analiza tych zagadnień zajmuje w tym wydaniu cały rozdział.
- **Pamięć podręczna.** Ma ona zasadnicze znaczenie przy projektowaniu procesorów o wysokiej wydajności, a projektowanie pamięci podręcznych staje się coraz bardziej złożone. W nowym wydaniu zagadnieniom tym poświęciłem cały rozdział.
- **Pamięć optyczna.** Informacje na ten temat zostały poszerzone i zaktualizowane.
- **Zaawansowana architektura DRAM.** Informacje na temat tej architektury zostały znacznie poszerzone, z uwzględnieniem SDRAM i RDRAM.
- **Wieloprzetwarzanie symetryczne, klastry i systemy niejednorodnego dostępu do pamięci.** Rozdział o organizacji równoległej został poszerzony i zaktualizowany.
- **Poszerzona pomoc dla wykładowców.** Jak już wspomniałem, książka ta zawiera teraz informacje pomocne do rozwiązywania różnych zadań. Więcej też jest informacji zamieszczonych w specjalnej witrynie WWW poświęconej tej książce.

Podziękowania

Nowe wydanie bardzo zyskało na wartości merytorycznej dzięki recenzjom wielu osób, które wielkodusznie poświęciły na nie swój czas i wiedzę. Oto osoby, które zaopiniowały całość (lub znaczną część) rękopisu: Willis King (University of Houston), Albert Heaney (California State University), A. S. Pandya (Florida Atlantic University), Yaser Khalifa (University of North Dakota) i Sanjeev Baskiyar (Auburn University).

Bardzo też jestem wdzięczny osobom, które dokonały szczegółowego przeglądu technicznego poszczególnych rozdziałów. Należą do nich: Nicole Kaiyan, Terje Mathisen, Daniel M. Pressel, Jeff Deifik, Bill Todd, Charlie Cassidy, Andy Isaacson, Alex Potemkin, Michael Spratte, Hatem Yassine, Grzegorz Mazur, Alan Lehotsky, Jonathan Hall, Sophie Wilson, Alan Alexander, David Vickers, Pete Smoot i Erik Seligman.

Profesor Cindy Norris z Appalachian State University brała udział w formułowaniu zadań domowych.

Profesor Miguel Angel Vega Rodriguez, prof. dr Juan Manuel Sánchez Pérez i prof. dr Juan Antonio Gómez Pulido – wszyscy z Universidad de Extremadura w Hiszpanii – opracowali problematykę SMPCache w podręczniku dla wykładowców i napisali przewodnik dla użytkowników tego pakietu.

Todd Bezenek z University of Wisconsin i James Stine z Lehigh University opracowali problematykę SimpleScalar w podręczniku dla wykładowców, a Todd napisał przewodnik dla użytkowników tego pakietu.

rzeniu. Zachowałem niemal takiej samej organizacji rozdziałów, ale znaczną część materiału przeredagowałem i dodałem sporo nowego. Oto niektóre, najbardziej godne uwagi zmiany:

- **Architektura IA-64 (Itanium).** W tej nowej architekturze są uwzględnione tak ważne koncepcje, jak predykatywne wykonywanie i spekulatywne ładowanie rozkazów. Opis i analiza tych zagadnień zajmuje w tym wydaniu cały rozdział.
- **Pamięć podręczna.** Ma ona zasadnicze znaczenie przy projektowaniu procesorów o wysokiej wydajności, a projektowanie pamięci podręcznych staje się coraz bardziej złożone. W nowym wydaniu zagadnieniom tym poświęcono cały rozdział.
- **Pamięć optyczna.** Informacje na ten temat zostały poszerzone i zaktualizowane.
- **Zaawansowana architektura DRAM.** Informacje na temat tej architektury zostały znacznie poszerzone, z uwzględnieniem SDRAM i RDRAM.
- **Wieloprzetwarzanie symetryczne, klastry i systemy niejednorodnego dostępu do pamięci.** Rozdział o organizacji równoległej został poszerzony i zaktualizowany.
- **Poszerzono pomoc dla wykładowców.** Jak już wspomniano, książka ta zawiera teraz informacje pomocne do rozwiązywania różnych zadań. Wiele z nich jest informacji zamieszczonych w specjalnej witrynie WWW poświęconej tej książce.

Podziękowania

Nowe wydanie bardzo zyskało na wartości merytorycznej dzięki recenzjom wielu osób, które wielkodusznie poświęciły na nie swój czas i wiedzę. Oto osoby, które zaopiniowały całość (lub znaczną część) rękopisu: Willis King (University of Houston), Albert Heaney (California State University), A. S. Pandya (Florida Atlantic University), Yaser Khalifa (University of North Dakota) i Sanjeev Baskiyar (Auburn University).

Bardzo też jestem wdzięczny osobom, które dokonały szczegółowego przeglądu technicznego pojedynczych rozdziałów. Należą do nich: Nicole Kaiyan, Terje Mathisen, Daniel M. Pressel, Jeff Deifik, Bill Todd, Charlie Cassidy, Andy Isaacson, Alex Potemkin, Michael Spratte, Hatem Yassine, Grzegorz Mazur, Alan Lebotsky, Jonathan Hall, Sophie Wilson, Andrzej Zawadzki, Brian Watson, Pete Smoot i Erik Seligman.

Profesor Cindy Norris z Appalachian State University brała udział w formułowaniu zadań domowych.

Profesor Miguel Angel Vega Rodriguez, profesor Juan Manuel Sánchez Pérez i profesor Juan Antonio Gómez Pulido – wszyscy z Universidad de Extremadura w Hiszpanii – opracowali problematykę SMPCache w podręczniku dla wykładowców i napisali przewodnik dla użytkowników tego pakietu.

Todd Bezenek z University of Wisconsin i James Stine z Lehigh University opracowali problematykę SimpleScalar w podręczniku dla wykładowców, a Todd napisał przewodnik dla użytkowników tego pakietu.

Część pierwsza
PRZEGLĄD

Problemy przedstawione w części pierwszej

W części pierwszej zamierzam przedstawić tło i kontekst pozostałych części tej książki. Zostaną przedstawione podstawowe koncepcje organizacji i architektury komputerów.

Tematy poszczególnych rozdziałów części pierwszej

Rozdział 1. Wprowadzenie

W rozdziale 1 jest wprowadzona koncepcja komputera jako systemu hierarchicznego. Komputer może być postrzegany jako pewna struktura złożona z zespołów, a jego działanie – jako zbiorowe działanie współpracujących ze sobą zespołów. Z kolei każdy zespół może być opisywany poprzez swoją wewnętrzną strukturę i działanie. Przedstawimy główne poziomy tego hierarchicznego obrazu. Pozostała część książki jest zorganizowana stosownie do tych poziomów, poczynając od najwyższego.

Rozdział 2. Ewolucja i wydajność komputerów

Rozdział 2 służy dwóm celom. Po pierwsze, zostanie ukazana historia techniki komputerowej, co umożliwi przedstawienie w przystępny i interesujący sposób podstawowych koncepcji organizacji i architektury komputerów. W rozdziale tym zostaną również przedstawione tendencje rozwoju tej techniki, które uczyniły wydajność głównym problemem przy projektowaniu systemów komputerowych, a także różne techniki i strategie, które służą do osiągania zrównoważonej wydajności.

Rozdział 1

Wprowadzenie

1.1. Organizacja i architektura
1.2. Struktura i działanie
Działanie
Struktura
1.3. Dlaczego warto studiować organizację i architekturę komputerów?
1.4. Zawartość książki
1.5. Zasoby Internetu i sieci WWW
Witryny WWW związane z tą książką
Inne witryny WWW
Grupy dyskusyjne USENET-u

Tematem tej książki jest struktura i działanie komputerów, celem moim zaś – możliwie jasne i kompletne przedstawienie natury i własności współczesnych systemów komputerowych. Z dwóch powodów zadanie to jest wyzwaniem.

Po pierwsze, istnieje ogromna różnorodność wyrobów, które mogą nosić miano komputera – od pojedynczego mikrokomputera kosztującego kilka dolarów do superkomputerów o wartości dziesiątków milionów dolarów. Różnorodność ta dotyczy nie tylko kosztu, ale także rozmiaru, wydajności i obszaru zastosowań. Po drugie, szybkie zmiany, które zawsze charakteryzowały technikę komputerową, zachodzą nadal i wcale nie słabną. Zmiany te obejmują wszystkie aspekty techniki komputerowej – od mającej podstawowe znaczenie technologii układów scalonych wykorzystywanej do budowy zespołów komputera, aż do coraz szerszego wykorzystania koncepcji organizacji równoległej przy łączeniu tych zespołów.

Pomimo tej różnorodności i tempa zmian w dziedzinie komputerów pewne podstawowe rozwiązania pozostają nadal aktualne. Z pewnością zastosowanie ich zależy od bieżącego stanu techniki oraz od założonej przez projektanta relacji między ceną a wydajnością, dlatego celem książki jest wnikliwe przedyskutowanie podstaw organizacji i architektury komputerów oraz powiązanie ich ze współczesnymi zagadnieniami projektowania komputerów. Rozdział ten ma charakter wprowadzenia i jest ujęty opisowo. Zawiera on przegląd pozostałej części książki.

1.1. Organizacja i architektura

Przy opisywaniu systemów komputerowych często czynione jest rozróżnienie między *architekturą* komputera a jego *organizacją*. Chociaż precyzyjne zdefiniowanie tych pojęć jest trudne, jednak istnieje zgoda co do zagadnień, których dotyczą (patrz np. [VRAN80], [SIEW82] i [BELL78a]).

Architektura komputera odnosi się do tych atrybutów systemu, które są widzialne dla programisty. Innymi słowy, atrybuty te mają bezpośredni wpływ na logiczne wykonywanie programu. *Organizacja komputera* odnosi się do jednostek operacyjnych i ich połączeń, które stanowią realizację specyfikacji typu architektury. Przykładami atrybutów architektury są: lista rozkazów, liczba bitów wykorzystywanych do prezentacji różnych typów danych (np. liczb czy znaków), mechanizmy wejścia-wyjścia oraz metody adresowania pamięci. Do atrybutów organizacyjnych należą rozwiązania sprzętowe niewidzialne dla programisty, takie jak sygnały sterujące, interfejsy między komputerem a urządzeniami peryferyjnymi oraz wykorzystywana technologia pamięci.

Na przykład to, czy w komputerze występuje rozkaz mnożenia, jest zagadnieniem projektowania architektury. Zagadnieniem organizacyjnym jest natomiast to, czy ten rozkaz będzie wykonywany przez specjalną jednostkę mnożącą, czy też przez wielokrotne wykorzystanie jednostki sumującej systemu. Decyzja organizacyjna może wynikać ze spodziewanej częstości wykorzystywania rozkazu mnożenia, ze względnej szybkości obu rozwiązań, a także z kosztu i fizycznych rozmiarów specjalnej jednostki mnożącej.

Historycznie, a także współcześnie, rozróżnienie między architekturą a organizacją jest ważne. Wielu producentów komputerów oferuje rodziny modeli o tej samej architekturze, różniące się jednak organizacją. W wyniku tego poszczególne modele w tej samej rodzinie mają różne ceny i parametry określające wydajność. Ponadto, architektura może przeżyć wiele lat, natomiast organizacja zmienia się wraz z technologią. Wybitnym przykładem obu tych zjawisk jest architektura Systemu 370 IBM. Architektura ta była po raz pierwszy wprowadzona w roku 1970 i obejmowała wiele modeli. Klient o umiarkowanych wymaganiach mógł kupić model tańszy, lecz wolniejszy. Jeśli jego wymagania wzrosły, mógł sięgnąć po droższy i szybszy model, nie rezygnując jednakże z dotychczas opracowanego oprogramowania. Przez całe lata IBM wprowadzał wiele nowych modeli wykorzystujących ulepszoną technologię w celu zastąpienia starszych modeli, oferując klientowi większą szybkość, niższy koszt lub i jedno, i drugie. Nowe modele zachowywały tę samą architekturę, co chroniło inwestycje poniesione przez klienta na oprogramowanie. Jest godne uwagi, że architektura Systemu 370, nieznacznie ulepszona, przeżyła do dzisiaj i nadal występuje w dużych komputerach wytwarzanych przez IBM.

W klasie systemów nazywanych mikrokomputerami zależność między architekturą a organizacją jest bardzo ścisła. Zmiany technologii nie tylko wpływają na organizację, ale także umożliwiają budowanie komputerów o potężniejszej i bogatszej architekturze. W przypadku małych maszyn wymagania dotyczące kompatybilności między generacjami są mniejsze. Możliwości różnorodnego łączenia decyzji organizacyjnych i architektonicznych są zatem większe. Intrygującym tego przykładem jest komputer o zredukowanej liście rozkazów (RISC), który omówimy w rozdz. 12.

W książce tej jest rozpatrywana zarówno organizacja, jak i architektura komputerów. Być może większy nacisk jest położony na organizację. Ponieważ jednak organizacja komputera musi być zaprojektowana w celu wdrożenia określonej architektury, dokładne przeanalizowanie organizacji wymaga również szczegółowego zbadania architektury.

1.2. Struktura i działanie

Komputer jest systemem złożonym; współczesne komputery zawierają miliony elementów elektronicznych. Jak więc można je prosto opisać? Kluczem jest rozpoznanie hierarchicznej struktury najbardziej złożonych systemów, w tym komputera [SIMO69]. System hierarchiczny jest układem wzajemnie powiązanych podsystemów, z których każdy również ma strukturę hierarchiczną, aż do osiągnięcia najniższego poziomu – podsystemu elementarnego.

Hierarchiczna struktura złożonych systemów ma podstawowe znaczenie zarówno dla projektowania, jak i opisu. W określonym momencie projektant operuje na jednym tylko, szczególnym poziomie systemu. Na każdym poziomie system obejmuje zespół składników i ich wzajemne zależności. Na każdym poziomie zachowanie zależy jedynie od uproszczonych, abstrakcyjnych własności następnego, niższego poziomu. Na każdym poziomie projektant zajmuje się strukturą i funkcjami składników systemu.

- **Struktura** to sposób wzajemnego powiązania składników.
- **Funkcje** określają działanie poszczególnych składników jako części struktury.

Istnieją dwie możliwości opisania systemu: zbudowanie kompletnego opisu począwszy od poziomu najniższego lub też rozpoczęcie od obrazu widzianego na poziomie najwyższym oraz dekompozycja systemu na części. Jest wiele dowodów na to, że podejście „od góry do dołu" jest jaśniejsze i najbardziej efektywne [WEIN75].

Podejście przyjęte w tej książce odzwierciedla ten właśnie punkt widzenia. System komputerowy będzie opisany „od góry do dołu". Rozpoczniemy od głównych składników systemu, opiszemy ich strukturę i funkcje, a następnie rozpatrzymy sukcesywnie niższe poziomy hierarchii. Pozostała część tego podrozdziału zawiera krótki opis tych zamierzeń.

Działanie

Zarówno struktura, jak i funkcjonowanie komputera są w zasadzie proste. Na rysunku 1.1 są pokazane podstawowe funkcje, które może realizować komputer. Ogólnie są to tylko cztery funkcje:

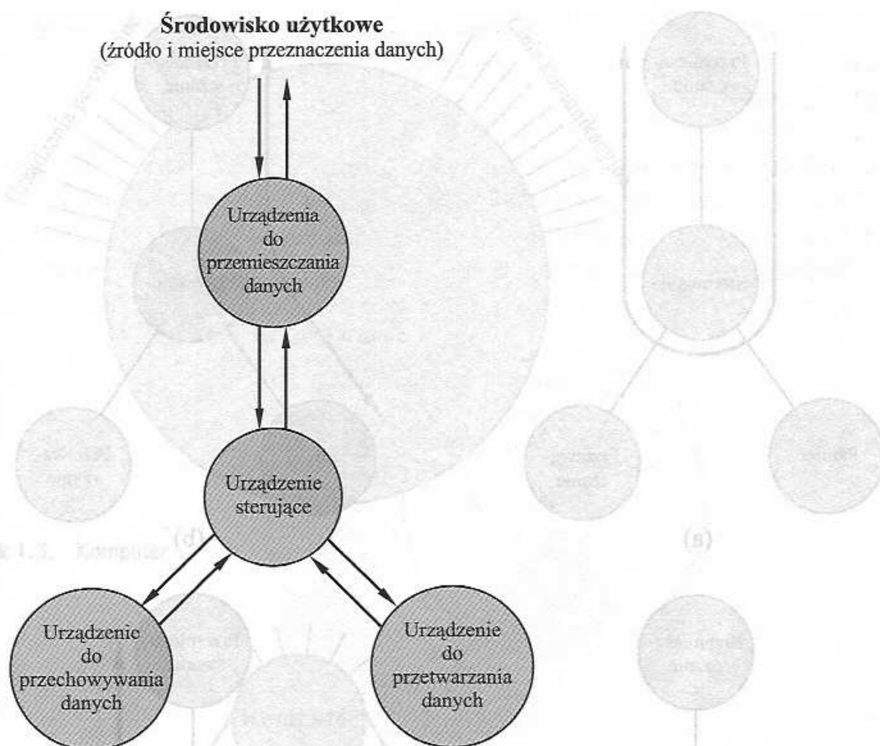
- przetwarzanie danych;
- przechowywanie danych;
- przenoszenie danych;
- sterowanie.

Oczywiście komputer musi móc *przetwarzać dane*. Dane mogą przybierać różne formy, a zakres wymagań odnoszących się do przetwarzania jest szeroki. Zobaczymy jednak, że istnieje tylko niewiele podstawowych metod, lub typów, przetwarzania danych.

Jest również bardzo ważne, aby komputer *przechowywał dane*. Nawet jeśli komputer przetwarza dane jedynie „w locie" (tzn. dane przychodzą, są przetwarzane i natychmiast wychodzą), musi on czasowo przechowywać chociażby te dane, które w danym momencie są przetwarzane. Występuje więc przynajmniej funkcja krótkotrwałego przechowywania. Równie ważne jest, aby komputer realizował funkcję długotrwałego przechowywania danych. Pliki danych są przechowywane w komputerze, co umożliwia ich późniejsze pobieranie i aktualizację.

Komputer musi być zdolny do *przenoszenia danych* między sobą a światem zewnętrznym. Otoczenie operacyjne komputera składa się z urządzeń, które są albo źródłami, albo odbiorcami danych. Jeśli dane są otrzymywane od urządzenia bezpośrednio połączonego z komputerem lub do niego dostarczane, to taki proces jest określany jako *proces wejścia-wyjścia*, a samo urządzenie nazywa się *peryferyjnym*. Jeśli dane są przenoszone na większe odległości, do odległego urządzenia lub od niego, to proces taki jest określany jako *transmisja danych*.

Musi wreszcie istnieć możliwość *sterowania* tymi trzema funkcjami. W ostateczności sterowanie to jest wykonywane przez osoby, które wydają komputerowi po-



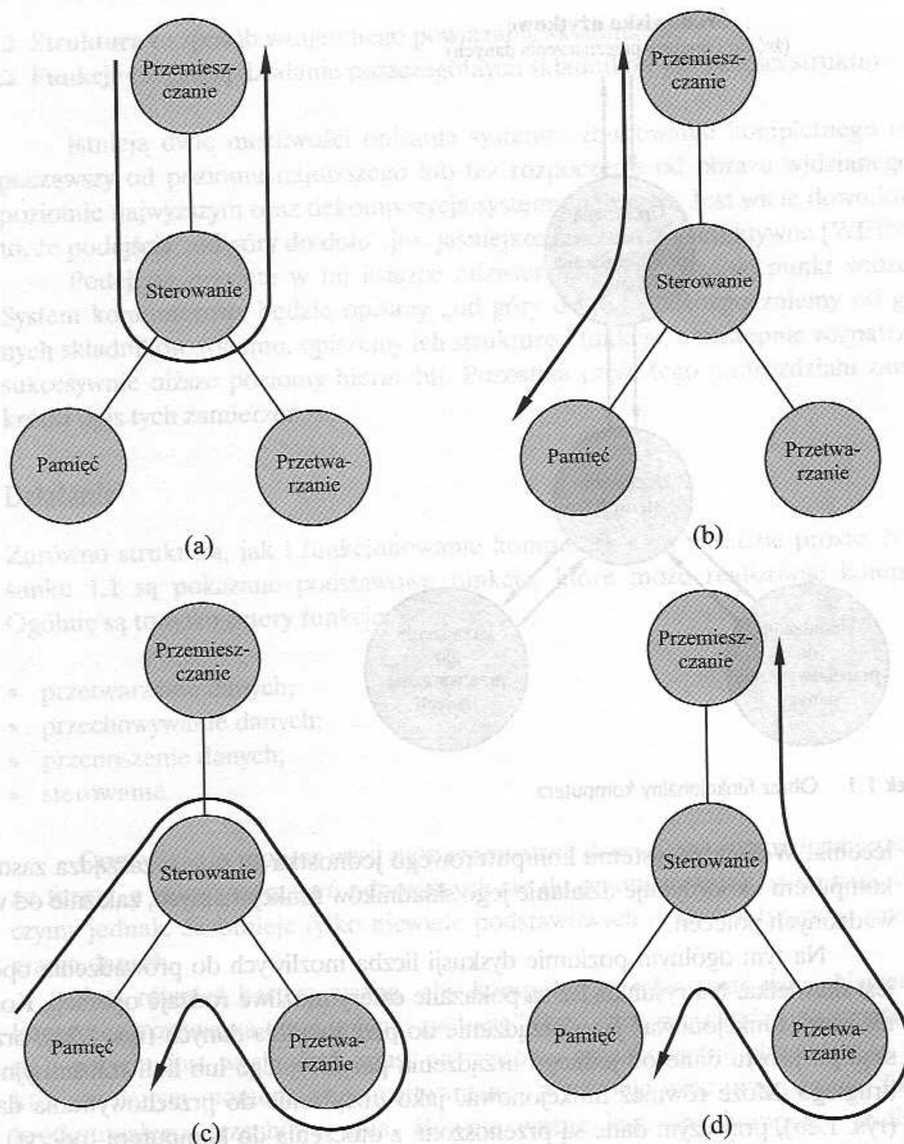
Rysunek 1.1. Obraz funkcjonalny komputera

lecenia. Wewnątrz systemu komputerowego jednostka sterująca zarządza zasobami komputera i koordynuje działanie jego składników funkcjonalnych, zależnie od wprowadzonych poleceń.

Na tym ogólnym poziomie dyskusji liczba możliwych do prowadzenia operacji jest niewielka. Na rysunku 1.2 są pokazane cztery możliwe rodzaje operacji. Komputer może funkcjonować jako urządzenie do przenoszenia danych (rys. 1.2a), przenosząc po prostu dane od jednego urządzenia peryferyjnego lub linii transmisyjnej do drugiego. Może również funkcjonować jako urządzenie do przechowywania danych (rys. 1.2b), przy czym dane są przenoszone z otoczenia do komputera (odczyt) i odwrotnie (zapis). Na dwóch ostatnich diagramach pokazano operacje obejmujące przetwarzanie danych – zarówno tych, które są przechowywane (rys. 1.2c), jak i będących w drodze między miejscem przechowywania a otoczeniem (rys. 1.2d).

Dotychczasowa dyskusja może wyglądać na absurdalnie ogólną. Z pewnością jest możliwe, nawet na najwyższym poziomie struktury komputera, wyróżnienie wielu funkcji. Zacytujmy tu jednak [SIEW82]:

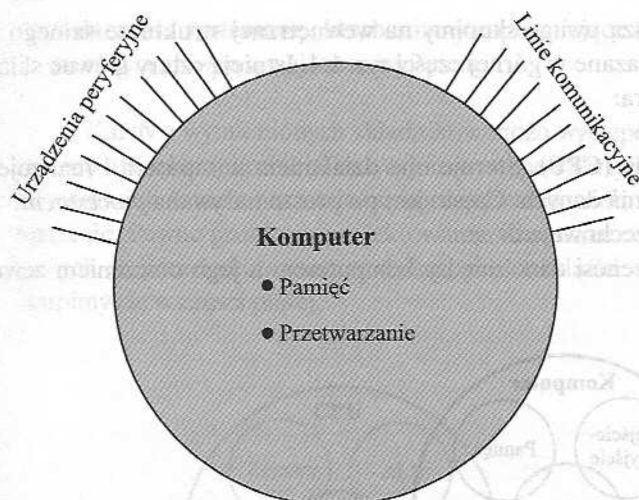
> Jest godne uwagi, jak nieznacznie kształtuje się strukturę komputera zależnie od funkcji, która ma być wykonywana. U źródła tego zjawiska leży z natury ogólne przeznaczenie komputerów, przy czym cała specjalizacja funkcjonalna następuje w czasie programowania, a nie podczas projektowania.



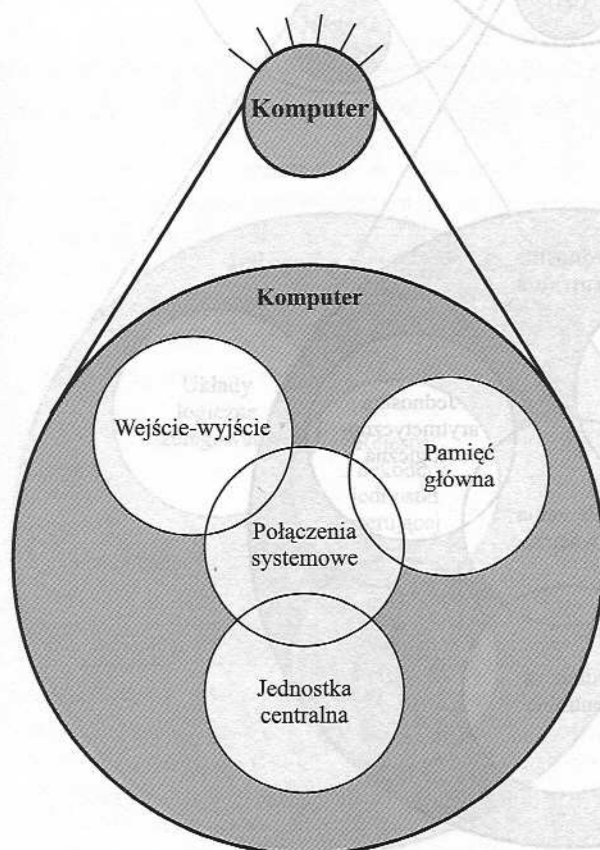
Rysunek 1.2. Operacje możliwe do wykonania przez komputer

Struktura

Na rysunku 1.3 jest pokazany możliwie najprostszy obraz komputera. Komputer jest urządzeniem, które jest w pewien sposób powiązane ze swoim otoczeniem zewnętrznym. Ogólnie rzecz biorąc, wszystkie jego powiązania z otoczeniem mogą być sklasyfikowane jako urządzenia peryferyjne lub linie transmisyjne. Będziemy musieli bliżej przedstawić oba typy tych powiązań.



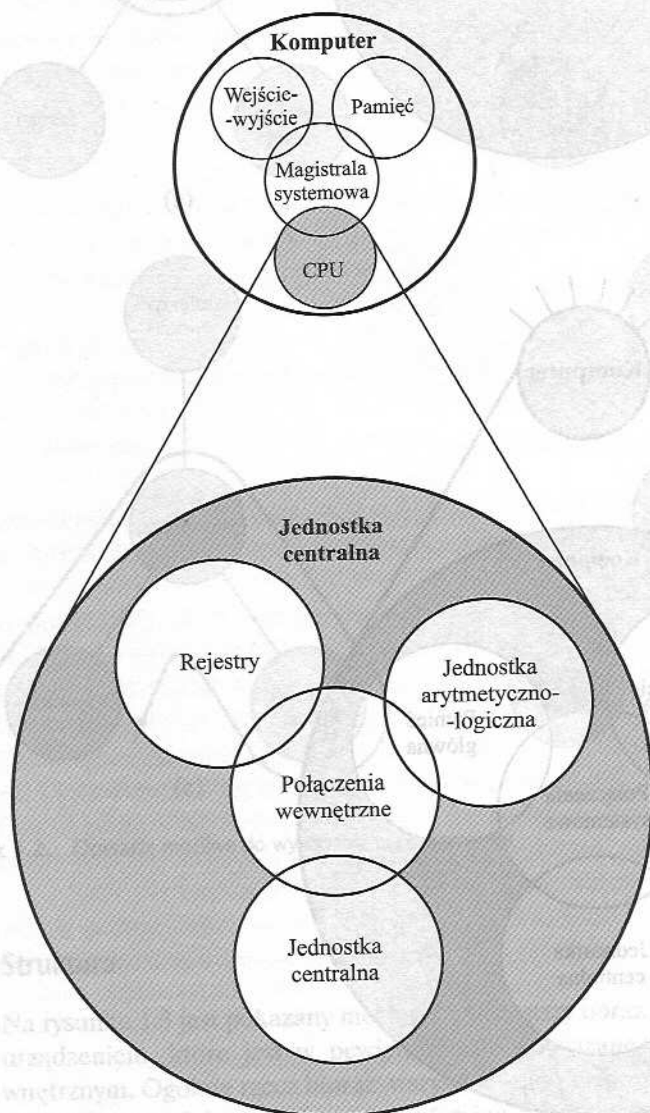
Rysunek 1.3. Komputer



Rysunek 1.4. Struktura komputera

Jednak największą uwagę skupimy na wewnętrznej strukturze samego komputera, co zostało pokazane w górnej części rys. 1.4. Istnieją cztery główne składniki strukturalne komputera:

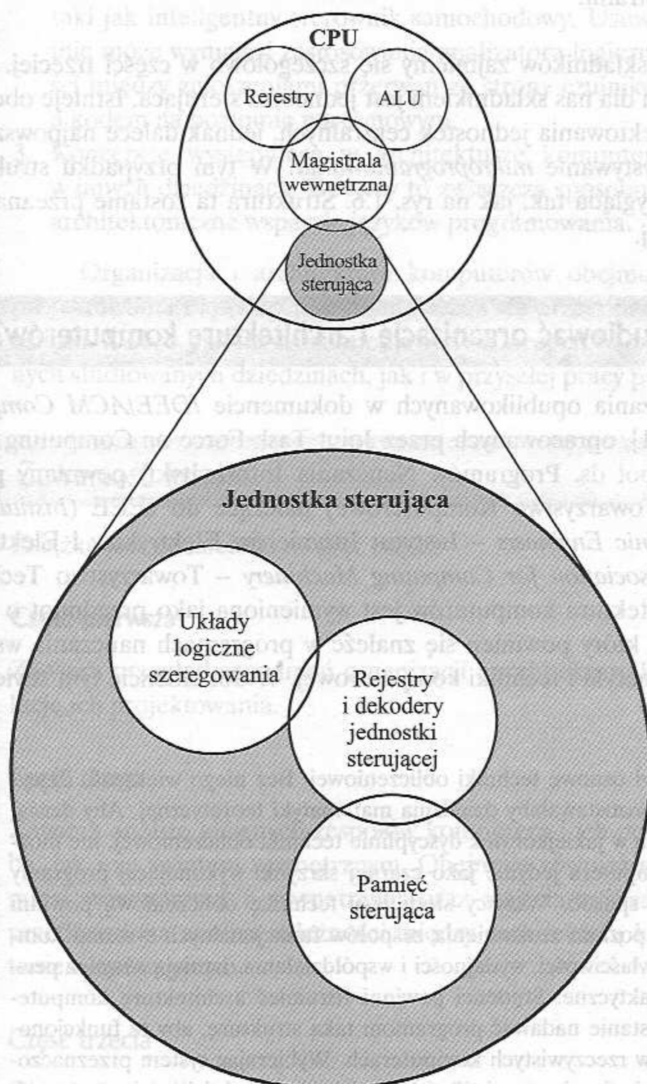
- **Jednostka centralna (CPU).** Steruje ona działaniem komputera i realizuje jego funkcję przetwarzania danych. Często jest po prostu nazywana *procesorem*.
- **Pamięć główna.** Przechowuje dane.
- **Wejście-wyjście.** Przenosi dane między komputerem a jego otoczeniem zewnętrznym.



Rysunek 1.5. Jednostka centralna (CPU)

- **Połączenia systemowe.** Mechanizmy zapewniające łączność między procesorem, pamięcią główną a wejściem-wyjściem.

Każdy z wymienionych składników może występować w komputerze pojedynczo lub w większej liczbie. Tradycyjnie występowała tylko jedna jednostka centralna. W ostatnich latach narastało wykorzystywanie wielu procesorów w pojedynczym systemie. Pewne problemy projektowania odnoszące się do systemów wieloprocesorowych są przedyskutowane w dalszej części książki. Na takich właśnie systemach skupimy się w części piątej.



Rysunek 1.6. Jednostka sterująca

Każdy z wymienionych składników omówimy szczegółowo w części drugiej. Dla naszych celów najbardziej interesującym i pod pewnymi względami najbardziej złożonym składnikiem jest jednostka centralna. Jej strukturę widać na rys. 1.5. Głównymi składnikami strukturalnymi procesora są:

- **Jednostka sterująca.** Steruje działaniem procesora i przez to komputera.
- **Jednostka arytmetyczno-logiczna (ALU).** Realizuje funkcję przetwarzania danych przez komputer.
- **Rejestry.** Realizują wewnętrzne przechowywanie danych w procesorze.
- **Połączenia procesora.** Mechanizm zapewniający łączność między jednostką sterującą, ALU i rejestrami.

Każdym z tych składników zajmiemy się szczegółowo w części trzeciej. Najbardziej interesującym dla nas składnikiem jest jednostka sterująca. Istnieje obecnie kilka podejść do projektowania jednostek centralnych, jednak dalece najpowszechniejszym jest wykorzystywanie *mikroprogramowania*. W tym przypadku struktura jednostki sterującej wygląda tak, jak na rys. 1.6. Struktura ta zostanie przeanalizowana w części czwartej.

1.3. Dlaczego warto studiować organizację i architekturę komputerów?

W programach nauczania opublikowanych w dokumencie *IDEE/ACM Computer Curricula 2001* [JTF01], opracowanych przez Joint Task Force on Computing Curricula (Wspólny Zespół ds. Programów Nauczania Informatyki) powołany przez Computer Society (Towarzystwo Komputerowe) należące do IEEE (*Institute of Electrical and Electronic Engineers* – Instytut Inżynierów Elektryków i Elektroników) oraz ACM (*Association for Computing Machinery* – Towarzystwo Techniki Komputerowej) architektura komputerów jest wymieniona jako przedmiot o podstawowym znaczeniu, który powinien się znaleźć w programach nauczania wszystkich studentów informatyki i techniki komputerowej. W dokumencie tym stwierdza się, co następuje:

> Komputer stanowi osnowę techniki obliczeniowej. Bez niego większość dyscyplin tej techniki pozostawałaby dziedziną matematyki teoretycznej. Aby dzisiaj być profesjonalistą w jakiejkolwiek dyscyplinie techniki obliczeniowej, nie można traktować komputera jedynie jako czarnej skrzynki wykonującej programy w jakiś magiczny sposób. Wszyscy studiujący technikę obliczeniową powinni osiągnąć pewien poziom zrozumienia zespołów funkcjonalnych systemu komputerowego, ich właściwości, wydajności i współdziałania. Istnieją również pewne implikacje praktyczne. Studenci powinni rozumieć architekturę komputerów, żeby byli w stanie nadawać programom taką strukturę, aby te funkcjonowały skuteczniej w rzeczywistych komputerach. Wybierając system przeznaczony do użytku, powinni oni rozumieć kompromisy między właściwościami różnych zespołów, takimi jak szybkość zegara jednostki centralnej a wielkość pamięci.

W [CLEM100] są podane następujące przykłady jako uzasadnienie studiowania architektury komputerów:

1. Załóżmy, że absolwent trafia do przemysłu i jest proszony o wybranie komputera, który byłby najbardziej efektywny pod względem kosztów w wielkiej organizacji. Do podjęcia takiej decyzji jest niezmiernie ważne zrozumienie implikacji poniesienia większych wydatków na takie rozwiązania, jak większa pamięć podręczna lub większa częstotliwość zegara procesora.
2. Procesory nie są stosowane wyłącznie w komputerach osobistych ani w serwerach, lecz często są wbudowywane w inne systemy. Projektant może programować w C procesor wbudowany w jakiś system pracujący w czasie rzeczywistym, taki jak inteligentny sterownik samochodowy. Usuwanie błędów w takim systemie może wymagać zastosowania analizatora logicznego, który ukazuje zależności między zgłoszeniami przerwań ze strony czujników umieszczonych w silniku a kodem na poziomie maszynowym.
3. Koncepcje występujące w architekturze komputerów znajdują zastosowanie w innych dziedzinach. Dotyczy to zwłaszcza sposobu, w jaki komputer zapewnia architektoniczne wsparcie języków programowania.

Organizacja i architektura komputerów obejmuje szeroki krąg zagadnień projektowania i koncepcji, o czym można się przekonać przeglądając spis treści tej książki. Dobre ogólne zrozumienie tych koncepcji będzie użyteczne zarówno w innych studiowanych dziedzinach, jak i w przyszłej pracy po ukończeniu studiów.

1.4. Zawartość książki

Książka jest podzielona na pięć części:

Część pierwsza

Zawiera przegląd zagadnień organizacji i architektury komputerów i ukazuje ewolucję ich projektowania.

Część druga

Zawiera analizę głównych zespołów komputera i ich połączeń, zarówno między sobą, jak i ze światem zewnętrznym. Obejmuje również szczegółowe omówienie pamięci wewnętrznej i zewnętrznej oraz wejścia-wyjścia. Na zakończenie została przedstawiona analiza zależności między architekturą komputera a używanym systemem operacyjnym.

Część trzecia

Obejmuje analizę wewnętrznej architektury i organizacji procesora. Część ta rozpoczyna się od poszerzonego przedstawienia arytmetyki komputerowej. Następnie

zajmujemy się architekturą listy rozkazów. Reszta tej części obejmuje strukturę i działanie procesora, łącznie z omówieniem rozwiązań o zredukowanej liście rozkazów i podejścia superskalarnego oraz szczegółowym przedstawieniem architektury IA-64.

Część czwarta

Zawiera omówienie wewnętrznej struktury jednostki sterującej procesora i mikroprogramowania.

Część piąta

Jest poświęcona organizacji równoległej, łącznie z wieloprzetwarzaniem symetrycznym i klastrami.

1.5. Zasoby Internetu i sieci WWW

W Internecie i w sieci WWW istnieje wiele zasobów wspomagających tę książkę i ułatwiających nadążanie za rozwojem tej dziedziny.

Witryny WWW związane z tą książką

Pod adresem WilliamStallings.com/COA6e.html została utworzona specjalna strona WWW związana z książką. Jej szczegółowy opis znajduje się na początku tej książki.

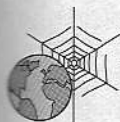
Errata do książki będzie publikowana i aktualizowana stosownie do potrzeb we wspomnianej witrynie. Proszę czytelników o informowanie mnie pocztą elektroniczną o wykrytych błędach. Erraty do moich pozostałych książek znajdują się w witrynie WilliamStallings.com.

Prowadzę również witrynę zawierającą zasoby przydatne dla studentów informatyki (*Computer Science Student Resource Site*) pod adresem WilliamStallings.com/StudentSupport.html; z założenia ma ona być źródłem dokumentów, informacji i łącz przydatnych dla studentów informatyki i osób zajmujących się nią zawodowo. Łącza zostały podzielone na cztery kategorie:

- **Matematyka.** Przypomnienie podstaw matematyki, elementarz analizy kolejkowania, elementarz systemów liczenia oraz łącza prowadzące do innych, przydatnych witryn poświęconych matematyce.
- **Porady.** Porady i wskazówki dotyczące rozwiązywania zadań domowych, pisania sprawozdań technicznych i przygotowywania prezentacji.
- **Zasoby informacyjne.** Łącza prowadzące do ważnych zbiorów artykułów, raportów technicznych i bibliografii.
- **Różne.** Przydatne dokumenty i łącza.

Inne witryny WWW

Istnieją liczne witryny WWW, w których znajdują się informacje związane z tematyką tej książki. W kolejnych rozdziałach odsyłacze do określonych witryn znajdują się w podrozdziałach „Polecana literatura i witryny WWW". Ponieważ lokalizatory URL witryn WWW ulegają częstym zmianom, nie włączałem ich do tej książki. Łącza prowadzące do wszystkich witryn wymienionych w książce można znaleźć w witrynie tej książki. W razie potrzeby będą dodawane inne łącza.



Następujące witryny WWW wiążą się z organizacją i architekturą komputerów:

- **WWW Computer Architecture Home Page.** Wyczerpujący skorowidz informacji istotnych dla osób zajmujących się architekturą komputerów, obejmujący m.in. zespoły i przedsięwzięcia z tej dziedziny, organizacje techniczne, literaturę, informacje o możliwości zatrudnienia i informacje handlowe.
- **CPU Info Center.** Informacje o poszczególnych procesorach wraz z artykułami technicznymi, informacjami o produktach i najnowszymi doniesieniami.
- **ACM Special Interest Group on Computer Architecture.** Informacje o działalności i publikacjach SIGARCH.
- **IEEE Technical Committee on Computer Architecture.** Kopie biuletynu TCAA.

Grupy dyskusyjne USENET-u

Istnieją grupy dyskusyjne USENET-u poświęcone niektórym aspektom organizacji i architektury komputerów. Podobnie jak w przypadku praktycznie wszystkich grup USENET-u, stosunek szumu do sygnału jest wysoki, warto jednak sprawdzić, czy któraś z nich nie okaże się przydatna. Oto grupy najbardziej odpowiadające tematyce książki:

- **comp.arch.** Ogólna grupa dyskusyjna zajmująca się architekturą komputerów. Często całkiem dobra.
- **comp.arch.arithmetic.** Zajmuje się algorytmami i standardami arytmetycznymi.
- **comp.arch.storage.** Dyskusja sięga od produktów poprzez technologie do problemów praktycznego użytkowania.
- **comp.parallel.** Obejmuje równoległe systemy komputerowe i ich zastosowania.

Inne witryny WWW

Istnieją liczne witryny WWW, w których znajdują się informacje związane z tematyką tej książki. W kolejnych rozdziałach odsyłacze do określonych witryn znajdują się w podrozdziałach „Polecana literatura i witryny WWW". Ponieważ lokalizatory URL witryn WWW ulegają częstym zmianom, nie włączałem ich do tej książki. Łącza prowadzące do wszystkich witryn wymienionych w książce można znaleźć w witrynie tej książki. W razie potrzeby będą dodawane inne łącza.

Następujące witryny WWW wiążą się z organizacją i architekturą komputerów:

- **WWW Computer Architecture Home Page.** Wyczerpujący skorowidz informacji istotnych dla osób zajmujących się architekturą komputerów, obejmujący m.in. zespoły i przedsięwzięcia z tej dziedziny, organizacje techniczne, literaturę, informacje o możliwości zatrudnienia i informacje handlowe.
- **CPU Info Center.** Informacje o poszczególnych procesorach wraz z artykułami technicznymi, informacjami o produktach i najnowszymi doniesieniami.
- **ACM Special Interest Group on Computer Architecture.** Informacje o działalności i publikacjach SIGARCH.
- **IEEE Technical Committee on Computer Architecture.** Kopie biuletynu TCAA.

Grupy dyskusyjne USENET-u

Istnieją grupy dyskusyjne USENET-u poświęcone niektórym aspektom organizacji i architektury komputerów. Podobnie jak w przypadku praktycznie wszystkich grup USENET-u stosunek szumu do sygnału jest wysoki, warto jednak sprawdzić, czy któraś z nich nie okaże się przydatna. Oto grupy najbardziej odpowiadające tematyce książki:

- **comp.arch.** Ogólna grupa dyskusyjna zajmująca się architekturą komputerów. Często całkiem dobra.
- **comp.arch.arithmetic.** Zajmuje się algorytmami i standardami arytmetycznymi.
- **comp.arch.storage.** Dyskusja sięga od produktów poprzez technologię do problemów praktycznego użytkowania.
- **comp.parallel.** Obejmuje równoległe systemy komputerowe i ich zastosowania.

Rozdział 2

Ewolucja i wydajność komputerów

2.1. Krótka historia komputerów

Pierwsza generacja: lampy próżniowe
Druga generacja: tranzystory
Trzecia generacja: układy scalone
Późniejsze generacje

2.2. Projektowanie zorientowane na wydajność

Szybkość mikroprocesora
Równoważenie wydajności

2.3. Ewolucja Pentium i PowerPC

Pentium
PowerPC

2.4. Polecana literatura i witryny WWW

2.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

PODSTAWOWE SPOSTRZEŻENIA

- Ewolucję komputerów charakteryzowały: wzrastająca szybkość procesora, malejące rozmiary zespołów, wzrastająca pojemność pamięci oraz rosnąca przepustowość i szybkość urządzeń wejścia-wyjścia.
- Jednym z czynników, które przesądzają o znacznym wzroście szybkości procesorów, jest zmniejszanie rozmiarów elementów mikroprocesorów; powoduje ono zmniejszanie odległości między elementami i tym samym zwiększa szybkość. Jednak w ostatnich latach istotny wzrost szybkości był wynikiem zmian organizacji procesorów, w tym intensywnego posługiwania się metodami kształtowania potoków rozkazów i równoległego ich wykonywania, a także stosowania techniki wykonywania spekulatywnego, dzięki której wstępnie są wykonywane rozkazy, które mogą być potrzebne. Wszystkie te techniki są projektowane tak, aby procesor był maksymalnie zajęty.
- Krytycznym problemem przy projektowaniu systemów komputerowych jest równoważenie wydajności różnych zespołów, tak aby zwiększenie wydajności w jednym obszarze nie było niwelowane przez niewystarczającą wydajność w innych obszarach. W szczególności szybkość procesora zwiększa się proporcjonalnie szybciej, niż jest redukowany czas dostępu do pamięci. Stosuje się wiele metod kompensujących to niedopasowanie, łącznie z pamięciami podręcznymi, szerszymi ścieżkami danych między pamięcią a procesorem oraz bardziej „inteligentnymi" mikroukładami pamięci.

Studiowanie komputerów rozpoczniemy od ich krótkiej historii. Historia ta – interesująca sama w sobie – umożliwia również przegląd struktury i działania komputerów. Następnie zajmiemy się problemem wydajności. Potrzeba zrównoważonego posługiwania się zasobami komputera stanowi kontekst, który zachowuje przydatność w całej książce. Na zakończenie przyjrzymy się pobieżnie ewolucji dwóch systemów, które w tej książce są najczęściej przywoływane jako przykłady: Pentium i PowerPC.

2.1. Krótka historia komputerów

Pierwsza generacja: lampy próżniowe

ENIAC

ENIAC (skrót od ang. *Electronic Numerical Integrator And Computer* – elektroniczne urządzenie numeryczne całkujące i liczące) został zaprojektowany i zbudowany pod nadzorem Johna Mauchly'ego i Johna Prespera Eckerta z University of Pennsylvania. Był pierwszym na świecie elektronicznym komputerem cyfrowym o ogólnym przeznaczeniu.

Przedsięwzięcie to było odpowiedzią na wojenne potrzeby USA w czasie drugiej wojny światowej. Wojskowe Laboratorium Badań Balistycznych (BRL), odpowiedzialne za opracowywanie tablic zasięgu i toru dla nowych broni, miało trudności z dostarczaniem tych tablic w rozsądnym czasie i przy zachowaniu wymaganej dokładności. Bez tych tablic nowe rodzaje broni i artyleria były bezużyteczne. BRL zatrudniało ponad 200 osób, w większości kobiet, które posługując się biurkowymi kalkulatorami rozwiązywały niezbędne równania balistyczne. Przygotowanie tablic dla pojedynczej broni zajmowało jednej osobie wiele godzin, a nawet dni.

Mauchly, który był profesorem elektrotechniki University of Pennsylvania, oraz Eckert, jeden z jego studentów, zaproponowali zbudowanie komputera o ogólnym przeznaczeniu przy wykorzystaniu lamp próżniowych, mając na uwadze zastosowanie go w BRL. W roku 1943 propozycja ta została zaakceptowana przez wojsko, po czym rozpoczęto prace nad maszyną ENIAC. Urządzenie było ogromne: ważyło 30 t, zajmowało ok. 1400 m^2 i zawierało ponad 18 000 lamp próżniowych. W czasie pracy zużywało moc 140 kW. Było znacznie szybsze niż jakikolwiek komputer elektromechaniczny, wykonując 5000 operacji dodawania na sekundę.

ENIAC był maszyną raczej dziesiętną niż binarną. Liczby były reprezentowane w postaci dziesiętnej i arytmetyka była realizowana w systemie dziesiętnym. Jego pamięć składała się z 20 „akumulatorów", z których każdy mógł przechowywać 10-cyfrową liczbę dziesiętną. Każda cyfra była reprezentowana przez pierścień złożony z 10 lamp próżniowych. W określonym czasie tylko jedna lampa próżniowa znajdowała się w stanie włączonym (ON), reprezentując jedną z 10 cyfr. Główną wadą ENIAC-a było to, że musiał on być programowany ręcznie przez ustawianie przełączników oraz wtykanie i wyjmowanie kabli.

ENIAC został ukończony w roku 1946, zbyt późno, by mógł być wykorzystany w wojnie. Zamiast tego, jego pierwszym zastosowaniem było przeprowadzenie serii złożonych obliczeń, które były pomocne przy określaniu wykonalności bomby wodorowej. Zastosowanie ENIAC-a do zadania innego niż to, dla którego został zbudowany, było demonstracją jego z natury ogólnego przeznaczenia. Rok 1946 zapowiedział więc erę komputerów elektronicznych, co było kulminacją całych lat wysiłków. ENIAC działał w laboratorium BRL do roku 1955, po czym został rozebrany.

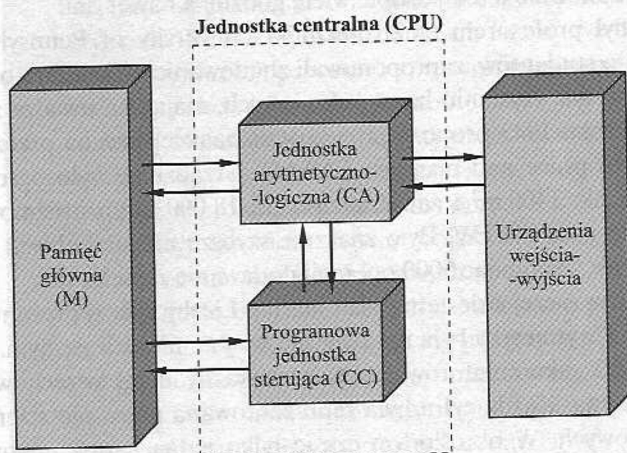
Maszyna von Neumanna

Jak już wspomnieliśmy, wprowadzanie i zmiana programów dla ENIAC-a było ekstremalnie żmudne. Proces programowania mógłby być ułatwiony, jeśliby program mógł być reprezentowany w postaci odpowiedniej do przechowywania w pamięci razem z danymi. Komputer pobierałby wtedy rozkazy przez odczytywanie ich z pamięci, a program mógłby być instalowany lub zmieniany przez zmianę zawartości części pamięci.

Idea ta, znana jako *koncepcja przechowywanego programu*, jest zwykle przypisywana projektantom ENIAC-a, głównie zaś matematykowi Johnowi von Neumannowi, który był konsultantem w tym przedsięwzięciu. Niemal w tym samym czasie koncepcja taka została opracowana przez Turinga. Po raz pierwszy idea ta została

opublikowana w roku 1945 w propozycji von Neumanna opracowania nowego komputera pod nazwą EDVAC (skrót od ang. *Electronic Discrete Variable Computer*).

W roku 1946 von Neumann i jego koledzy rozpoczęli projektowanie nowego komputera wykorzystującego program przechowywany w pamięci. Miało to miejsce w Princeton Institute for Advanced Studies, a komputer określono skrótem IAS. Komputer IAS, chociaż nie został ukończony do roku 1952, był prototypem wszystkich następnych komputerów o ogólnym przeznaczeniu.



Rysunek 2.1. Struktura komputera IAS

Na rysunku 2.1 jest pokazana ogólna struktura komputera IAS. Jego składnikami są:

- Pamięć główna, w której są przechowywane zarówno dane, jak i rozkazy.
- Jednostka arytmetyczno-logiczna (ALU) mogąca wykonywać działania na danych binarnych.
- Jednostka sterująca, która interpretuje rozkazy z pamięci i powoduje ich wykonanie.
- Urządzenia wejścia-wyjścia, których pracą kieruje jednostka sterująca.

Struktura ta została przedstawiona we wcześniejszej propozycji von Neumanna. Warto ją tutaj zacytować [VONM45]:

> 2.2. **Po pierwsze**, ponieważ urządzenie to jest przede wszystkim komputerem, najczęściej będzie wykonywało elementarne operacje arytmetyczne – dodawanie, odejmowanie, mnożenie i dzielenie (+, –, ×, :). Jest więc rozsądne, że powinno mieć wyspecjalizowane „organy" do wykonywania tych operacji.
>
> Należy jednak zauważyć, że chociaż powyższa zasada jako taka brzmi rozsądnie, to szczegółowy sposób jej realizacji wymaga głębokiego zastanowienia... W każdym przypadku *centralna*, *arytmetyczna* część urządzenia będzie prawdopodobnie musiała istnieć, co oznacza występowanie *pierwszej specyficznej części komputera*: *CA*.

2.3. **Po drugie**, logiczne sterowanie urządzeniem, to znaczy odpowiednie szeregowanie jego operacji, może być najefektywniej realizowane przez centralny organ sterujący. Jeśli urządzenie ma być *elastyczne*, to znaczy możliwie *uniwersalne*, należy rozróżniać specyficzne rozkazy związane z określonym problemem i ogólne „organy" sterujące, dbające o wykonanie tych rozkazów – czymkolwiek by one nie były. Te pierwsze muszą być w jakiś sposób przechowywane; te drugie – reprezentowane przez określone działające części urządzenia. Przez *sterowanie centralne* rozumiemy tylko tę ostatnią funkcję, a „organy", które ją realizują, tworzą *drugą, specyficzną część urządzenia*: *CC*.

2.4. **Po trzecie**, (a) jakiekolwiek urządzenie, które ma wykonywać długie i skomplikowane sekwencje działań (w szczególności obliczenia), musi mieć odpowiednio dużą pamięć...

(b) Rozkazów kierujących rozwiązywaniem skomplikowanego problemu może być bardzo dużo, zwłaszcza wtedy, kiedy kod jest przypadkowy (a tak jest w większości wypadków). Muszą one być pamiętane...

W każdej sytuacji całkowita pamięć stanowi *trzecią, specyficzną część urządzenia*: *M*.

2.6. Trzy specyficzne części CA, CC (razem C) oraz M odpowiadają neuronom skojarzeniowym w systemie nerwowym człowieka. Pozostają do przedyskutowania równoważniki neuronów *sensorycznych* (doprowadzających) i *motorycznych* (odprowadzających). Są to organy *wejścia* i *wyjścia* naszego urządzenia.

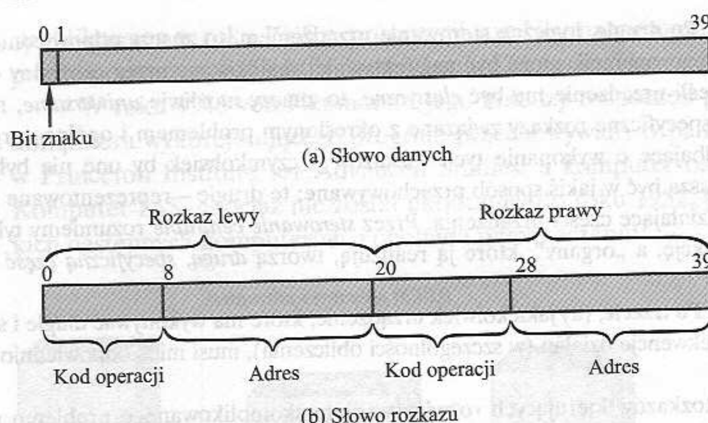
Urządzenie musi mieć możliwość utrzymywania kontaktu z wejściem i wyjściem za pomocą specjalistycznego narzędzia (porównaj 1.2). Narzędzie to będzie nazywane *zewnętrznym narzędziem rejestrującym urządzenia*: *R*...

2.7. **Po czwarte**, urządzenie musi być wyposażone w organy przenoszące ...informację z R do swoich specyficznych części C i M. „Organy" te stanowią jego wejście, a więc *czwartą, specyficzną część*: *I*. Zobaczymy, że najlepiej jest dokonywać wszystkich przeniesień z R (poprzez I) do M, a nigdy bezpośrednio do C...

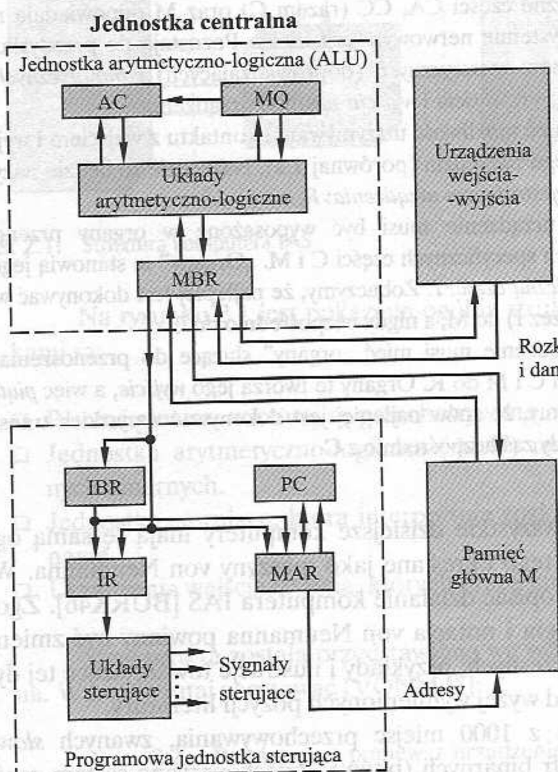
2.8. **Po piąte**, urządzenie musi mieć „organy" służące do przenoszenia ...ze swoich specyficznych części C i M do R. Organy te tworzą jego *wyjście*, a więc *piątą specyficzną część*: *O*. Zobaczymy, że znów najlepiej jest dokonywać wszystkich transferów z M (poprzez O) do R, nigdy zaś bezpośrednio z C...

Z rzadkimi wyjątkami wszystkie dzisiejsze komputery mają tę samą ogólną strukturę i funkcję, są wobec tego określane jako maszyny von Neumanna. Warto więc w tym momencie krótko opisać działanie komputera IAS [BURK46]. Zgodnie z pracą [HAYE98] terminologia i notacja von Neumanna powinny być zmienione stosownie do wymagań współczesnych; przykłady i ilustracje towarzyszące tej dyskusji są zgodne z ostatnią spośród wyżej wymienionych pozycji literatury.

Pamięć IAS składa się z 1000 miejsc przechowywania, zwanych *słowami*, z których każde zawiera 40 cyfr binarnych (bitów). Przechowywane są tam zarówno dane, jak i rozkazy. Liczby muszą więc być prezentowane w postaci binarnej, a każdy rozkaz również musi być w kodzie binarnym. Formaty te są pokazane na rys. 2.2. Każda liczba jest reprezentowana przez bit znaku i wartość 39-bitową. Słowo może także zawierać dwa rozkazy 20-bitowe, przy czym każdy rozkaz składa się z 8-bitowego kodu operacji określającego operację, która ma być realizowana, oraz 12-bitowego adresu określającego jedno ze słów w pamięci (ponumerowanych od 0 do 999).



Rysunek 2.2. Formaty pamięci IAS



Rysunek 2.3. Rozszerzona struktura komputera IAS

Jednostka sterująca uruchamia IAS, pobierając rozkaz z pamięci i wykonując go – jeden rozkaz w określonym momencie. W celu wyjaśnienia tego działania jest potrzebny bardziej szczegółowy obraz struktury, pokazany na rys. 2.3. Na rysunku

widać, że zarówno jednostka sterująca, jak i ALU zawierają miejsca pamięci zwane *rejestrami*, określane następująco:

- **Rejestr buforowy pamięci** (*memory buffer register* – MBR). Zawiera słowo, które ma być przechowywane w pamięci, lub też jest wykorzystywany do pobierania słów z pamięci.
- **Rejestr adresowy pamięci** (*memory address register* – MAR). Określa adres w pamięci dotyczący słowa, które ma być zapisane w rejestrze MBR lub z niego odczytane.
- **Rejestr rozkazów** (*instruction register* – IR). Zawiera 8-bitowy kod operacyjny rozkazu, który jest wykonywany.
- **Buforowy rejestr rozkazów** (*instruction buffer register* – IBR). Jest wykorzystywany do czasowego przechowywania podręcznego rozkazu pochodzącego ze słowa w pamięci.
- **Licznik programu** (*program counter* – PC). Zawiera adres następnej pary rozkazów, która ma być pobrana z pamięci.
- **Akumulator (AC)** i **rejestr mnożenia-dzielenia** (*multiplier-quotier* – MQ). Wykorzystywane do czasowego przechowywania argumentów i wyników operacji prowadzonych przez ALU. Na przykład wynikiem mnożenia dwóch liczb 40-bitowych jest liczba 80-bitowa; 40 najbardziej znaczących bitów przechowuje się w akumulatorze (AC), a najmniej znaczące w rejestrze MQ.

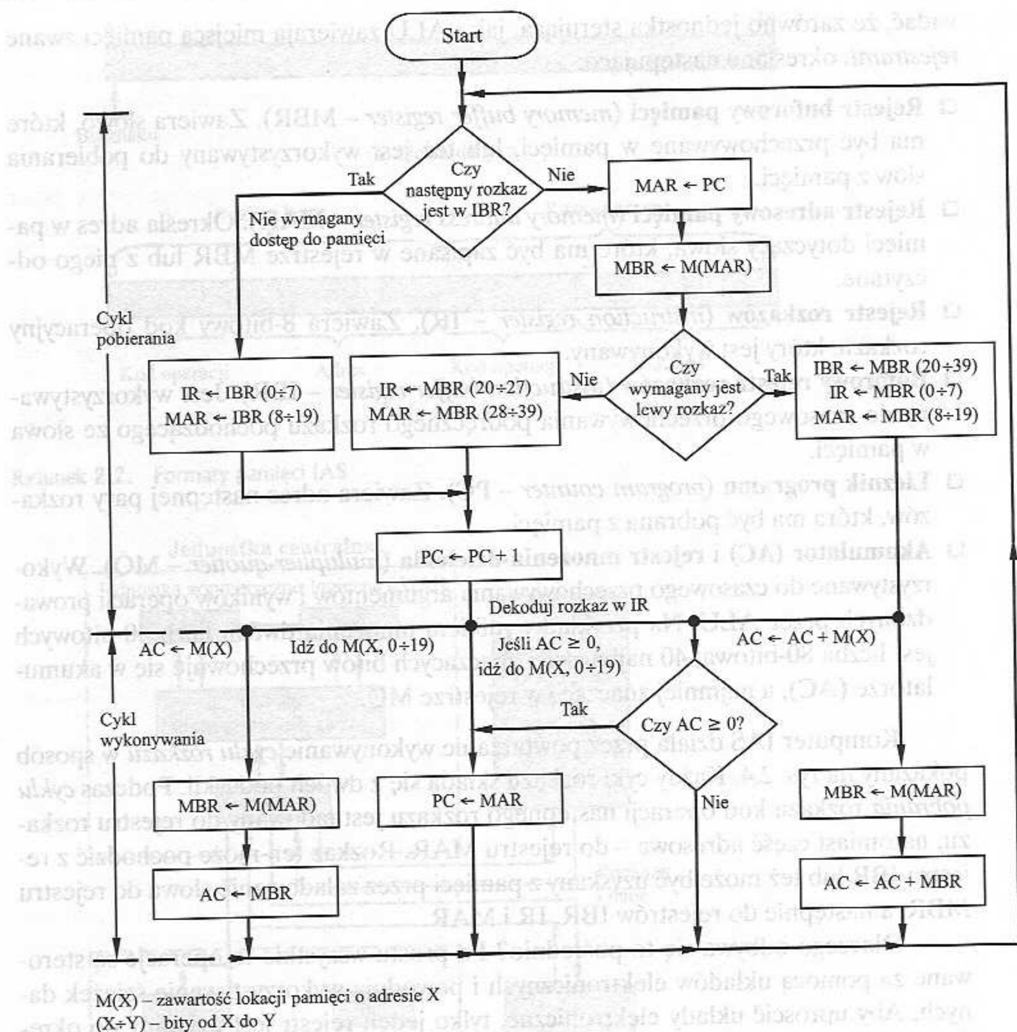
Komputer IAS działa przez powtarzalne wykonywanie *cyklu rozkazu* w sposób pokazany na rys. 2.4. Każdy cykl rozkazu składa się z dwóch podcykli. Podczas *cyklu pobrania* rozkazu kod operacji następnego rozkazu jest ładowany do rejestru rozkazu, natomiast część adresowa – do rejestru MAR. Rozkaz ten może pochodzić z rejestru IBR lub też może być uzyskany z pamięci przez załadowanie słowa do rejestru MBR, a następnie do rejestrów IBR, IR i MAR.

Dlaczego odbywa się to pośrednio? Po prostu wszystkie te operacje są sterowane za pomocą układów elektronicznych i powodują wykorzystywanie ścieżek danych. Aby uprościć układy elektroniczne, tylko jeden rejestr jest używany do określania adresu w pamięci zarówno przy odczycie, jak i przy zapisie. Rolę źródła i miejsca przeznaczenia również gra tylko jeden rejestr.

Gdy kod operacji znajdzie się w rejestrze IR, realizowany jest *cykl wykonywania*. Układy sterujące rozpoznają kod operacji i wykonują rozkaz, wysyłając odpowiednie sygnały sterujące, które powodują, że przenoszone są dane lub ALU wykonuje operację.

Komputer IAS ma łącznie 21 rozkazów. Są one wymienione w tabeli 2.1. Można je zakwalifikować do następujących grup:

- **Przenoszenie danych.** Należą tu rozkazy powodujące przesyłanie danych między pamięcią a rejestrami ALU lub między dwoma rejestrami ALU.
- **Rozgałęzienia bezwarunkowe.** Zwykle jednostka sterująca wykonuje szeregowo rozkazy pochodzące z pamięci. Sekwencja rozkazów może być zmieniona przez rozkaz skoku. Umożliwia to wykonywanie operacji powtarzalnych.



Rysunek 2.4. Częściowa sieć działań komputera IAS

- **Rozgałęzienia warunkowe.** Skoki mogą być wykonywane zależnie od pewnego warunku, co oznacza występowanie punktów decyzyjnych.
- **Arytmetyka.** Operacje wykonywane przez ALU.
- **Modyfikowanie adresu.** Należą tu rozkazy, które umożliwiają obliczanie adresów w ALU, a następnie wstawianie ich do rozkazów przechowywanych w pamięci. Dzięki temu program uzyskuje znaczną elastyczność adresowania.

W tabeli 2.1 są przedstawione rozkazy w postaci symbolicznej, łatwej do odczytania. W rzeczywistości każdy rozkaz musi być zgodny z formatem przedstawionym na rys. 2.2b. Część stanowiąca kod operacji (pierwszych 8 bitów) służy do ustalenia, który z 21 rozkazów ma być wykonywany. Część adresowa (pozostałych 12 bitów) służy do ustalenia, które z 1000 miejsc pamięci ma być zaangażowane w wykonywanie rozkazu.

Tabela 2.1. Lista rozkazów IAS

Rodzaj rozkazu	Kod operacji	Reprezentacja symboliczna	Opis
Transfer danych	00001010	LOAD MQ	Przenieś zawartość rejestru MQ do akumulatora AC.
	00001001	LOAD MQ,M(X)	Przenieś zawartość lokacji pamięci X do MQ.
	00100001	STOR M(X)	Przenieś zawartość akumulatora do lokacji pamięci X.
	00000001	LOAD M(X)	Przenieś M(X) do akumulatora.
	00000010	LOAD – M(X)	Przenieś – M(X) do akumulatora.
	00000011	LOAD |M(X)|	Przenieś wartość bezwzględną M(X) do akumulatora.
	00000100	LOAD – |M(X)|	Przenieś – |M(X)| do akumulatora.
Rozgałęzienie bezwarunkowe	00001101	JUMP M(X, 0÷19)	Pobierz następny rozkaz z lewej połowy M(X).
	00001110	JUMP M(X, 20÷39)	Pobierz następny rozkaz z prawej połowy M(X).
Rozgałęzienie warunkowe	00001111	JUMP + M(X, 0÷19)	Jeśli liczba w akumulatorze jest nieujemna, pobierz następny rozkaz z lewej połowy M(X).
	00010000	JUMP + M(X, 20÷39)	Jeśli liczba w akumulatorze jest nieujemna, pobierz następny rozkaz z prawej połowy M(X).
Arytmetyczne	00000101	ADD M(X)	Dodaj M(X) do akumulatora; umieść wynik w akumulatorze.
	00000111	ADD |M(X)|	Dodaj |M(X)| do akumulatora; umieść wynik w akumulatorze.
	00000110	SUB M(X)	Odejmij M(X) od akumulatora; umieść wynik w akumulatorze.
	00001000	SUB |M(X)|	Odejmij |M(X)| od akumulatora; umieść resztę w akumulatorze.
	00001011	MUL M(X)	Pomnóż M(X) przez MQ; umieść najbardziej znaczące bity wyniku w akumulatorze, a najmniej znaczące w MQ.
	00001100	DIV M(X)	Podziel zawartość akumulatora przez M(X); umieść iloraz w MQ, a resztę w akumulatorze.
	00010101	LSH	Pomnóż zawartość akumulatora przez 2, tzn. przesuń w lewo o jedną pozycję.
	00010010	RSH	Podziel zawartość akumulatora przez 2, tzn. przesuń w prawo o jedną pozycję.
Modyfikacja adresu	00010010	STOR M(X, 8÷19)	Zamień położone po lewej pole adresowe M(X) na 12 bitów akumulatora położonych po prawej.
	00010011	STOR M(X, 28÷39)	Zamień położone po prawej pole adresowe M(X) na 12 bitów akumulatora położonych po prawej .

Na rysunku 2.4 widać kilka przykładowych rozkazów wykonywanych przez jednostkę sterującą. Zauważmy, że każda operacja wymaga kilku kroków. Niektóre z nich są całkiem złożone. Operacja mnożenia wymaga 39 podoperacji, po jednej dla każdej pozycji bitowej z wyjątkiem bitu znaku!

Komputery komercyjne

W latach pięćdziesiątych narodził się przemysł komputerowy, przy czym dwie firmy – Sperry i IBM – zdominowały rynek.

W roku 1947 Eckert i Mauchly utworzyli Eckert-Mauchly Computer Corporation, aby produkować komputery w celach komercyjnych. Ich pierwszą udaną maszyną był UNIVAC I (skrót od ang. *Universal Automatic Computer*), który został wykorzystany przez Bureau of the Census (Biuro Spisu Ludności) do obliczeń roku 1950. Eckert-Mauchly Computer Corporation stała się częścią oddziału UNIVAC firmy Sperry-Rand Corporation, która rozpoczęła budowę szeregu maszyn pochodnych.

UNIVAC I był pierwszym udanym komputerem komercyjnym. Był przeznaczony, jak wynika z jego nazwy, zarówno do zastosowań naukowych, jak i komercyjnych. W pierwszym artykule opisującym system tego komputera wymieniono następujące przykładowe zadania, które może on realizować: macierzowe rachunki algebraiczne, problemy statystyczne, obliczenia składek dla firmy ubezpieczającej na życie oraz problemy logistyczne.

UNIVAC II, o większej pojemności pamięci i większej wydajności niż UNIVAC I, powstał w późnych latach pięćdziesiątych i stanowił ilustrację kilku tendencji, które pozostały charakterystyczne dla przemysłu komputerowego. Po pierwsze, postęp techniczny umożliwia firmom budowanie coraz większych i potężniejszych komputerów. Po drugie, każda z firm próbuje zapewnić *kompatybilność nowych maszyn w stosunku do starszych.* Oznacza to, że programy napisane dla starszych maszyn mogą być stosowane w nowej maszynie. Strategia ta wynika z nadziei na zachowanie klientów; jeśli klient zdecyduje się na kupno nowej maszyny, zechce ją nabyć raczej w tej samej firmie, aby uniknąć ewentualnych strat środków zainwestowanych w programy.

Oddział UNIVAC zapoczątkował także rozwój serii komputerów 1100, które miały stanowić jego źródło utrzymania. Seria ta jest ilustracją rozróżnienia, które występowało przez pewien czas. Pierwszy model (UNIVAC 1103) oraz jego następcy przez wiele lat byli przeznaczeni do zastosowań naukowych, obejmujących długie i złożone obliczenia. Inne firmy skoncentrowały się na zastosowaniach w biznesie, polegających na przetwarzaniu dużych ilości danych tekstowych. Podział ten w większości zanikł, był jednak zauważalny przez wiele lat.

IBM, który był największym producentem urządzeń do przetwarzania wykorzystujących karty perforowane, wyprodukował swój pierwszy komputer elektroniczny z przechowywanym programem – model 701 – w roku 1953. Komputer 701 był pierwotnie przeznaczony do zastosowań naukowych [BASH81]. W roku 1955 IBM wprowadził komputer 702, mający wiele cech sprzętowych, które odpowiadały zastosowaniom w biznesie. Były to pierwsze modele z długiej serii komputerów 700/7000, która umożliwiła firmie IBM osiągnięcie pozycji dominującego producenta komputerów.

Druga generacja: tranzystory

Pierwsza wielka zmiana w komputerach elektronicznych nadeszła wraz z zastąpieniem lampy próżniowej przez tranzystor. Tranzystor jest mniejszy, tańszy i wydziela mniej ciepła niż lampa próżniowa, a może być używany do budowy komputerów podobnie jak lampa. W przeciwieństwie do lampy próżniowej, która wymaga drutów, płytek metalowych i obudowy szklanej, tranzystor jest *elementem półprzewodnikowym* wykonanym z krzemu.

Tranzystor został wynaleziony w Bell Laboratories w roku 1947, a w latach pięćdziesiątych wywołał rewolucję elektroniczną. Dopiero jednak w późnych latach pięćdziesiątych całkowicie tranzystorowe komputery stały się osiągalne handlowo. I znów IBM nie był pierwszą firmą, która wykorzystała tę nową technologię. NCR oraz (z większym sukcesem) RCA były pierwszymi firmami dostarczającymi małe maszyny tranzystorowe. IBM wkroczył niedługo po nich z serią 7000.

Wykorzystanie tranzystora oznaczało początek *drugiej generacji komputerów*. Szeroko przyjęte jest klasyfikowanie komputerów na generacje wynikające z podstawowej technologii sprzętu (tab. 2.2). Każda następna generacja wyróżnia się większą szybkością, większą pojemnością pamięci i mniejszymi rozmiarami niż poprzednia.

Tabela 2.2. Generacje komputerów

Generacja	Lata (w przybliżeniu)	Technologia	Typowa szybkość (operacji na sekundę)
1	1946–1957	lampa próżniowa	40 000
2	1958–1964	tranzystor	200 000
3	1965–1971	mały i średni stopień scalenia	1 000 000
4	1972–1977	duży stopień scalenia	10 000 000
5	1978–	bardzo duży stopień scalenia	100 000 000

Są również jeszcze inne zmiany. W drugiej generacji wprowadzono bardziej złożone jednostki arytmetyczno-logiczne oraz sterujące, zastosowano ulepszone języki programowania, a także rozpoczęto dostarczanie wraz z komputerem *oprogramowania systemowego*.

Wraz z drugą generacją pojawiła się firma Digital Equipment Corporation (DEC). DEC została założona w roku 1957 i w tym samym roku wyprodukowała pierwszy komputer – PDP-1. Ten komputer i ta firma dali początek minikomputerom, które stały się tak ważne w trzeciej generacji.

IBM 7094

Od wprowadzenia serii 700 w roku 1952 do wprowadzenia ostatniego modelu rodziny 7000 w roku 1964 linia wyrobów IBM przechodziła ewolucję typową dla wyrobów komputerowych. Kolejne maszyny miały coraz większą wydajność, większą pojemność oraz (lub) niższy koszt.

Tabela 2.3. Przykładowe modele serii 700/7000 IBM

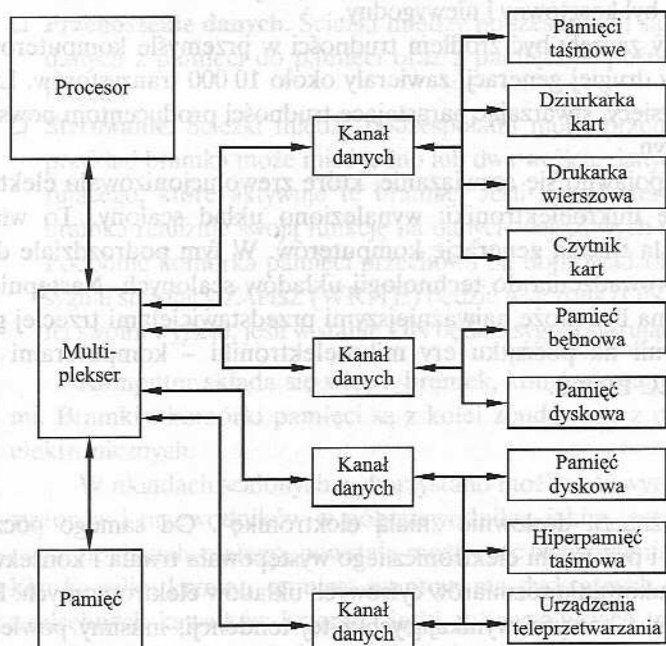
Numer modelu	Pierwsza dostawa	Technologia CPU	Technologia pamięci	Czas cyklu [μs]	Rozmiar pamięci [KB]	Liczba kodów operacji	Liczba rejestrów modyfikacji	Układowe rozwiązanie obliczeń zmiennopozycyjnych	Nakładanie się wejścia-wyjścia (kanały)	Nakładanie się pobierania rozkazów	Szybkość (odniesiona do 701)
701	1952	lampy próżniowe	lampy elektrostatyczne	30	2÷4	24	0	nie	nie	nie	1
704	1955	lampy próżniowe	rdzenie	12	4÷32	80	3	tak	nie	nie	2,5
709	1958	lampy próżniowe	rdzenie	12	32	140	3	tak	tak	nie	4
7090	1960	tranzystory	rdzenie	2,18	32	169	3	tak	tak	nie	25
7094 I	1962	tranzystory	rdzenie	2	32	185	7	tak (podwójna dokładność)	tak	tak	30
7094 II	1964	tranzystory	rdzenie	1,4	32	185	7	tak (podwójna dokładność)	tak	tak	50

Dane zawarte w tabeli 2.3 ilustrują tę tendencję. Rozmiar pamięci głównej, liczony w wielokrotnościach 2^{10} słów 36-bitowych, wzrósł z 2 K (1 K = 2^{10}) do 32 K słów, podczas gdy czas dostępu do słowa w pamięci, to znaczy *czas cyklu pamięci*, zmniejszył się od 30 µs do 1,4 µs. Liczba kodów operacji wzrosła od umiarkowanych 24 aż do 185. W ostatniej kolumnie jest podana względna szybkość działania procesora.

Zwiększenie szybkości osiągnięto w wyniku rozwoju elektroniki (np. tranzystory działają szybciej niż lampy próżniowe) oraz przez wzrost złożoności układów. Na przykład w IBM 7094 zastosowano pomocniczy rejestr rozkazów (*Instruction Backup Register* – IBR), wykorzystywany do przechowywania (buforowania) następnego rozkazu. Na rozkaz pobrania jednostka sterująca pobiera dwa sąsiednie słowa z pamięci. Z wyjątkiem występowania rozkazu skoku, co nie jest zbyt częste, jednostka sterująca potrzebuje zaledwie połowę cyklu rozkazu na dostęp do rozkazu w pamięci. To tak zwane wstępne pobieranie (lub inaczej pobieranie z wyprzedzeniem, ang. *prefetching*) znacznie redukuje przeciętny czas cyklu rozkazu.

Pozostałe kolumny tabeli 2.3 staną się zrozumiałe w dalszym ciągu naszych rozważań.

Na rysunku 2.5 jest pokazana rozszerzona (z wieloma urządzeniami peryferyjnymi) konfiguracja IBM 7094, która jest reprezentatywna dla komputerów drugiej generacji [BELL71a]. Warto odnotować kilka różnic w stosunku do komputera IAS. Najważniejszą z nich jest użycie *kanałów danych*. Kanał danych jest niezależnym modułem wejścia-wyjścia z własnym procesorem i własną listą rozkazów. W systemie komputerowym wyposażonym w takie urządzenia procesor nie wykonuje



Rysunek 2.5. Konfiguracja IBM 7094

szczegółowych rozkazów wejścia-wyjścia. Rozkazy takie są przechowywane w pamięci głównej, po czym są wykonywane przez specjalny procesor w samym kanale danych. Procesor inicjuje przesyłanie danych z wejścia-wyjścia przez wysłanie sygnału sterującego do kanału danych, powodując wykonanie przez niego sekwencji rozkazów zawartych w pamięci. Kanał danych realizuje swoje zadanie niezależnie od procesora, sygnalizuje tylko procesorowi zakończenie operacji. W takim rozwiązaniu procesor jest odciążony od wykonywania części przetwarzania.

Inną nową cechą jest występowanie *multipleksera*, który stanowi centralny punkt zbieżny kanałów danych, procesora i pamięci. Multiplekser szereguje dostęp procesora i kanałów danych do pamięci, pozwalając tym urządzeniom na działanie niezależne.

Trzecia generacja: układy scalone

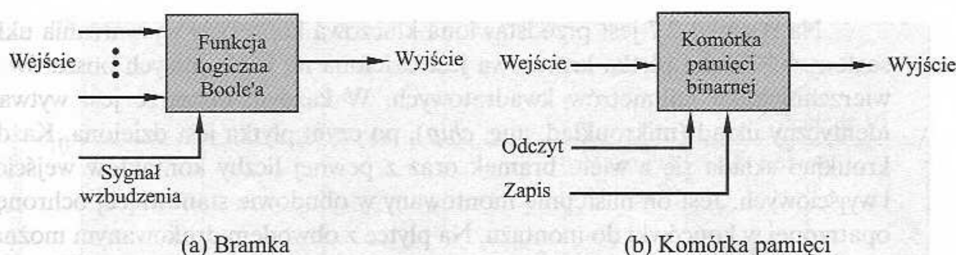
Pojedynczy, zamknięty we własnej obudowie tranzystor jest nazywany podzespołem (elementem) *dyskretnym*. Przez lata pięćdziesiąte i wczesne sześćdziesiąte urządzenia elektroniczne były złożone w większości z podzespołów dyskretnych – tranzystorów, rezystorów, kondensatorów itd. Podzespoły dyskretne były produkowane oddzielnie, montowane w obudowy oraz lutowane lub mocowane w inny sposób na płytkach drukowanych, które z kolei były instalowane w komputerach, oscyloskopach i innych urządzeniach elektronicznych. Mała metalowa obudowa w kształcie walca zawierająca mikroskopijny kawałek krzemu, zwana tranzystorem, musiała być przylutowana do obwodu drukowanego. Cały proces wytwarzania, od tranzystora do zmontowanej płytki, był kosztowny i niewygodny.

Powyższe fakty zaczęły być źródłem trudności w przemyśle komputerowym. Wczesne komputery drugiej generacji zawierały około 10 000 tranzystorów. Liczba ta urosła do setek tysięcy, stwarzając narastające trudności producentom nowszych, potężniejszych maszyn.

W roku 1958 pojawiło się rozwiązanie, które zrewolucjonizowało elektronikę i rozpoczęło erę mikroelektroniki: wynaleziono układ scalony. To właśnie układ scalony określa trzecią generację komputerów. W tym podrozdziale dokonamy krótkiego wprowadzenia do technologii układów scalonych. Następnie zapoznamy się z dwoma być może najważniejszymi przedstawicielami trzeciej generacji, wprowadzonymi na początku ery mikroelektroniki – komputerami IBM System 360 oraz DEC PDP-8.

Mikroelektronika

Mikroelektronika oznacza dosłownie „małą elektronikę". Od samego początku elektroniki cyfrowej i przemysłu elektronicznego występowała trwała i konsekwentna tendencja do redukowania rozmiarów cyfrowych układów elektronicznych. Przed zbadaniem następstw i korzyści wynikających z tej tendencji, musimy powiedzieć więcej o naturze elektroniki cyfrowej. Bardziej szczegółową dyskusję można znaleźć w dodatku A.



Rysunek 2.6. Podstawowe elementy komputera

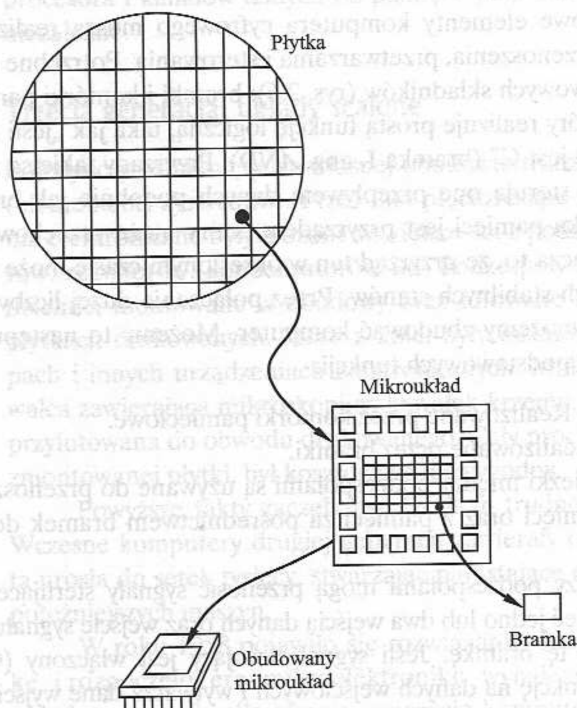
Jak wiemy, podstawowe elementy komputera cyfrowego muszą realizować funkcje przechowywania, przenoszenia, przetwarzania i sterowania. Potrzebne są do tego tylko dwa typy podstawowych składników (rys. 2.6): bramki i komórki pamięci. Bramka jest przyrządem, który realizuje prostą funkcję logiczną, taką jak „jeśli A i B są prawdziwe, to prawdziwe jest C" (bramka I, ang. *AND*). Przyrządy takie są nazywane bramkami, ponieważ sterują one przepływem danych podobnie jak bramki (śluzy) w kanałach. Komórka pamięci jest przyrządem, który może przechowywać pojedynczy bit danych; oznacza to, że przyrząd ten w określonym czasie może znajdować się w jednym z dwóch stabilnych stanów. Przez połączenie dużej liczby tych podstawowych przyrządów możemy zbudować komputer. Możemy to następująco odnieść do naszych czterech podstawowych funkcji:

- **Przechowywanie danych.** Realizowane przez komórki pamięciowe.
- **Przetwarzanie danych.** Realizowane przez bramki.
- **Przenoszenie danych.** Ścieżki między podzespołami są używane do przenoszenia danych z pamięci do pamięci oraz z pamięci za pośrednictwem bramek do pamięci.
- **Sterowanie.** Ścieżki między podzespołami mogą przenosić sygnały sterujące. Na przykład bramka może mieć jedno lub dwa wejścia danych oraz wejście sygnału sterującego, które aktywuje tę bramkę. Jeśli sygnał sterujący jest włączony (ON), bramka realizuje swoją funkcję na danych wejściowych i wytwarza dane wyjściowe. Podobnie komórka pamięci przechowa bit doprowadzony do swojego wejścia, jeśli sygnał sterujący ZAPISZ (WRITE) będzie włączony (ON), natomiast umieści ten bit na swoim wyjściu, jeśli w stanie ON będzie sygnał sterujący CZYTAJ (READ).

Komputer składa się więc z bramek, komórek pamięci i połączeń między nimi. Bramki i komórki pamięci są z kolei zbudowane z pojedynczych podzespołów elektronicznych.

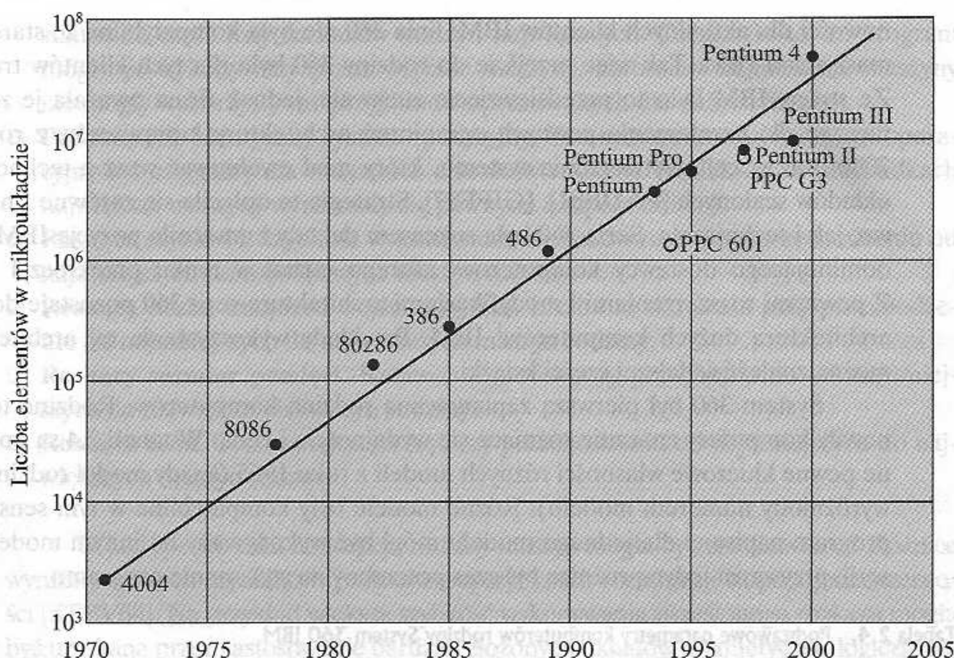
W układach scalonych wykorzystano możliwość wytwarzania tranzystorów, rezystorów i przewodników w półprzewodniku, jakim jest krzem. Dzięki rozwojowi technologii ciała stałego, powstała możliwość wytworzenia całego układu w niewielkim kawałku krzemu, zamiast montowania dyskretnych podzespołów wykonanych z odrębnych kawałków krzemu. Setki, a nawet tysiące tranzystorów można jednocześnie wytworzyć w pojedynczej płytce krzemowej. Równie ważne jest, że tranzystory te mogą być połączone w procesie metalizacji, tworząc układy.

Na rysunku 2.7 jest przedstawiona kluczowa koncepcja wytwarzania układów scalonych. Cienka płytka krzemowa jest dzielona na wiele małych obszarów o powierzchni kilku milimetrów kwadratowych. W każdym obszarze jest wytwarzany identyczny układ (mikroukład, ang. *chip*), po czym płytka jest dzielona. Każdy mikroukład składa się z wielu bramek oraz z pewnej liczby kontaktów wejściowych i wyjściowych. Jest on następnie montowany w obudowie stanowiącej ochronę i zaopatrzonej w końcówki do montażu. Na płytce z obwodem drukowanym można następnie zmontować pewną liczbę takich obudów w celu wytworzenia większych i bardziej złożonych układów.



Rysunek 2.7. Związek między płytką, mikroukładem i bramką

Początkowo można było niezawodnie wytwarzać jedynie kilka bramek lub komórek pamięciowych w jednym mikroukładzie. Te wczesne układy scalone noszą miano układów o *małym stopniu scalenia* (*small-scale integration* – SSI). W miarę upływu czasu stało się możliwe upakowywanie coraz większej ilości elementów w tym samym mikroukładzie. Ten wzrost gęstości jest zilustrowany na rys. 2.8; jest to jedna z najbardziej godnych uwagi tendencji technologicznych, jakie kiedykolwiek zarejestrowano. Rysunek ten odzwierciedla słynne prawo Moore'a, które zostało zaproponowane w roku 1965 przez Gordona Moore'a, współzałożyciela firmy Intel [MOOR65]. Moore zauważył, że liczba tranzystorów, które mogą być umieszczone w jednym mikroukładzie, podwajała się corocznie; poprawnie przewidział, że tempo to zostanie zachowane w najbliższej przyszłości. Ku zaskoczeniu wielu osób, w tym samego Moore'a, równie



Rysunek 2.8. Wzrost liczby tranzystorów w procesorze

szybkie zmiany trwały nadal, krok po kroku i dekada po dekadzie. W latach siedemdziesiątych tempo to zmniejszyło się na tyle, że podwajanie następowało co 18 miesięcy, jednak ustabilizowało się na tym poziomie aż do teraz.

Konsekwencje prawa Moore'a są doniosłe:

1. W tym okresie szybkiego wzrostu gęstości upakowania koszt mikroukładu pozostawał praktycznie niezmieniony. Oznacza to, że koszt układów logicznych i pamięciowych komputera malał bardzo szybko.
2. Ponieważ elementy logiczne i pamięciowe są umieszczane coraz bliżej siebie w coraz gęściej upakowanych mikroukładach, połączenia elektryczne uległy skróceniu, co doprowadziło do zwiększenia szybkości działania.
3. Zmniejszają się rozmiary komputerów, dzięki czemu można je wygodnie umieszczać w najróżniejszych środowiskach.
4. Maleją wymagania odnoszące się do zużycia mocy i chłodzenia.
5. Wzajemne połączenia elementów w układzie scalonym są znacznie bardziej niezawodne niż połączenia lutowane. Gdy w każdym mikroukładzie mieści się więcej układów, mniej jest połączeń między mikroukładami.

System 360 IBM

W roku 1964 IBM panował na rynku komputerowym, produkując maszyny serii 7000. W tym samym roku firma zaanonsowała System 360, nową rodzinę komputerów. Chociaż ta zapowiedź nie była niespodzianką, zawierała pewne nieprzyjemne

nowości dla aktualnych klientów IBM: linia 360 nie była kompatybilna ze starszymi maszynami IBM. Tak więc przejście do rodziny 360 było dla tych klientów trudne. Ze strony IBM było to przedsięwzięcie zuchwałe, jednak firma uważała je za konieczne dla przełamania pewnych ograniczeń architektury komputerów z rodziny 7000 oraz w celu wytworzenia systemu, który miał ewoluować wraz z technologią układów scalonych [PADE81], [GIFF87]. Strategia ta opłaciła się zarówno finansowo, jak i technicznie. Seria 360 była sukcesem dekady i umocniła pozycję IBM jako dominującego dostawcy komputerów, którego udział w rynku przekroczył 70%. Z pewnymi rozszerzeniami i modyfikacjami architektura serii 360 pozostaje do dziś architekturą dużych komputerów[1] IBM. Przykłady wykorzystania tej architektury można znaleźć w dalszej części książki.

System 360 był pierwszą zaplanowaną rodziną komputerów. Rodzinę tę stanowiły komputery znacznie różniące się wydajnością i ceną. W tabeli 2.4 są pokazane pewne kluczowe własności różnych modeli z roku 1965 (każdy model rodziny jest wyróżniony numerem modelu). Różne modele były kompatybilne w tym sensie, że program napisany dla jednego modelu mógł być wykonywany na innym modelu tej serii, przy czym jedyną różnicą był czas potrzebny na wykonanie programu.

Tabela 2.4. Podstawowe parametry komputerów rodziny System 360 IBM

Parametr	Model 30	Model 40	Model 50	Model 65	Model 75
Maksymalna pojemność pamięci (w bajtach)	64 K	256 K	256 K	512 K	512 K
Szybkość pobierania danych z pamięci [MB/s]	0,5	0,8	2,0	8,0	16,0
Czas cyklu procesora [μs]	1,0	0,625	0,5	0,25	0,2
Względna szybkość	1	3,5	10	21	50
Maksymalna liczba kanałów danych	3	3	4	6	6
Maksymalna szybkość przesyłania danych przez jeden kanał [KB/s]	250	400	800	1250	1250

Koncepcja rodziny kompatybilnych komputerów była zarówno nowa, jak i bardzo udana. Klient o umiarkowanych wymaganiach i budżecie mógł zacząć od względnie taniego modelu 30. Później, jeśli potrzeby klienta wzrosły, możliwe było zainteresowanie się szybszą maszyną o większej pamięci bez utraty inwestycji poniesionych dotychczas na oprogramowanie. Własności rodziny 360 są następujące:

- **Podobna lub identyczna lista rozkazów.** W wielu przypadkach dokładnie ta sama lista rozkazów służyła wszystkim członkom rodziny. Program wykonywany na jednej maszynie mógł być zatem wykonywany na dowolnej innej. W pewnych przypadkach najmniejsze maszyny szeregu posługiwały się listą rozkazów, która

[1] Wyrażenie *duże komputery* oznacza tutaj komputery określane w języku angielskim jako *mainframe*: komputery o większych rozmiarach i większej wydajności, nie będące jednak tzw. superkomputerami. Zwykle obsługują one wielkie bazy danych, mają rozbudowane urządzenia wejścia-wyjścia i są używane w centralnych zakładach przetwarzania danych.

stanowiła część listy rozkazów maszyn największych. Oznaczało to, że program mógł być wykonywany przez większe maszyny, natomiast najmniejsze maszyny nie mogły wykorzystywać programów opracowanych dla dużych maszyn.

- **Podobny lub identyczny system operacyjny.** Ten sam podstawowy system operacyjny był osiągalny dla wszystkich członków rodziny. W niektórych przypadkach największe maszyny wyróżniały się pewnymi uzupełnieniami.
- **Rosnąca szybkość.** Szybkość wykonywania rozkazów rosła przy przechodzeniu od najmniejszych do największych maszyn rodziny.
- **Rosnąca liczba urządzeń wejścia-wyjścia.** Liczba ta wzrastała przy przechodzeniu od najmniejszych do największych maszyn rodziny.
- **Rosnący rozmiar pamięci.** Rozmiar ten rósł przy przechodzeniu od najmniejszych do największych maszyn rodziny.
- **Rosnąca cena.** Cena zwiększała się przy przechodzeniu od najmniejszych do największych maszyn rodziny.

W jaki sposób taka koncepcja rodziny maszyn mogła być zrealizowana? Różnice wynikały z trzech czynników: podstawowej szybkości, rozmiaru i stopnia jednoczesności [STEV64]. Na przykład większa szybkość wykonywania określonego rozkazu mogła być uzyskana przez zastosowanie bardziej złożonych układów arytmetyczno-logicznych, pozwalających na równoległe wykonywanie podoperacji. Innym sposobem zwiększenia szybkości było zwiększenie szerokości ścieżki danych między pamięcią główną a procesorem. W modelu 30 tylko 1 bajt (8 bitów) mógł być w określonym momencie pobierany z pamięci, podczas gdy w modelu 70 możliwe było pobieranie 8 bajtów.

System 360 nie tylko wyznaczał przyszły kierunek rozwoju IBM, lecz także miał ogromny wpływ na cały przemysł. Wiele jego cech stało się normą dla dużych komputerów.

DEC PDP-8

W tym samym roku, w którym IBM wprowadził swój system 360, pojawiło się inne doniosłe opracowanie: komputer PDP-8 firmy Digital Equipment Corporation (DEC). Podczas gdy przeciętny komputer wymagał klimatyzowanego pomieszczenia, PDP-8 (ochrzczony w kręgach przemysłowych jako minikomputer na podobieństwo modnych wówczas minispódniczek) był dostatecznie mały, aby można go było umieścić na stole laboratoryjnym lub wbudować do innego urządzenia. Nie mógł on dokonać wszystkiego tego, co duża maszyna, jednak przy cenie 16 000 dolarów był tani na tyle, że każdy technik laboratoryjny mógł dysponować takim urządzeniem. Dla kontrastu, komputery Systemu 360 wprowadzone na rynek zaledwie kilka miesięcy wcześniej kosztowały setki tysięcy dolarów.

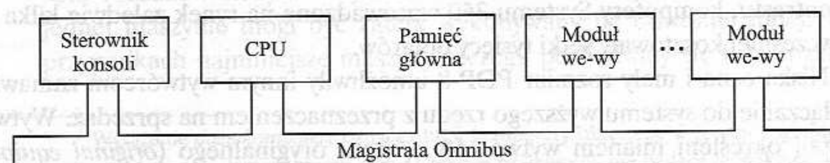
Niska cena i mały rozmiar PDP-8 umożliwiły innym wytwórcom zamawianie go i włączanie do systemu wyższego rzędu z przeznaczeniem na sprzedaż. Wytwórcy ci zostali określeni mianem wytwórców sprzętu oryginalnego (*original equipment manufacturers* – OEM), zaś rynek OEM stał się i pozostaje nadal głównym segmentem rynku komputerowego.

Komputer PDP-8 natychmiast stał się hitem, dzięki czemu firma DEC zrobiła fortunę. Maszyna ta, a także inne modele rodziny PDP-8 (patrz tab. 2.5), osiągnęła status produkcyjny zarezerwowany uprzednio dla komputerów IBM. W ciągu następnych 12 lat sprzedano około 50 000 maszyn. Jak podaje oficjalna historia firmy DEC, PDP-8 „określił koncepcję minikomputerów, torując drogę przemysłowi o wartości wielu miliardów dolarów". Pozwolił również na osiągnięcie przez DEC pozycji pierwszego dostawcy minikomputerów. Kiedy zaś PDP-8 osiągnął kres swojej drogi życiowej, DEC stała się drugim po IBM producentem komputerów.

Tabela 2.5. Ewolucja PDP-8 [VOEL88]

Model	Pierwsza sprzedaż	Koszt procesora + pamięć 4 K słów 12-bitowych (1000 USD)	Szybkość pobierania danych z pamięci [słów/μs]	Objętość (stóp sześciennych)	Innowacje i udoskonalenia
PDP-8	4/65	16,2	1,26	8,0	automatyczne wykonywanie połączeń owijanych (*wire-wrapping*)
PDP-8/5	9/66	8,79	0,08	3,2	wdrożenie rozkazów szeregowych
PDP-8/1	4/68	11,6	1,34	8,0	układy o średnim stopniu scalenia
PDP-8/L	11/68	7,0	1,26	2,0	mniejsza obudowa
PDP-8/E	3/71	4,99	1,52	1,8	Omnibus
PDP-8/M	6/72	3,69	1,52	1,8	o połowę mniejsza obudowa i mniej gniazd w porównaniu z 8/E
PDP-8/A	1/75	2,6	1,34	1,2	pamięć półprzewodnikowa; procesor zmiennopozycyjny

W przeciwieństwie do architektury opartej na centralnym przełączaniu (rys. 2.5) wykorzystanej przez IBM w systemach 700/7000 i 360, w ostatnich modelach PDP-8 zastosowano strukturę, która praktycznie stała się uniwersalna dla minikomputerów i mikrokomputerów: strukturę magistralową. Jest ona przedstawiona na rys. 2.9. Magistrala PDP-8, nazwana *Omnibusem*, składa się z 96 oddzielnych ścieżek sygnałów, używanych do przenoszenia sygnałów sterowania, adresu i danych. Ponieważ wszystkie urządzenia systemu używają wspólnego systemu ścieżek sygnało-



Rysunek 2.9. Struktura magistralowa komputera PDP-8

wych, ich wykorzystywaniem musi sterować procesor. Architektura ta jest wysoce elastyczna, gdyż umożliwia dołączanie różnych modułów do magistrali w celu tworzenia różnych konfiguracji.

Późniejsze generacje

Powyżej trzeciej generacji zgoda w sprawie definiowania generacji komputerów nie jest już pełna. Jak wynika z tabeli 2.2, powstały już generacje czwarta i piąta, wykorzystujące postępy w technologii układów scalonych. Wprowadzenie dużego stopnia scalenia (*large-scale integration* – LSI) pozwoliło na umieszczenie ponad 1000 elementów w pojedynczym mikroukładzie. Bardzo wielki stopień scalenia (*very-large-scale integration* – VLSI) oznaczał ponad 10 000 elementów w mikrostrukturze, a współczesne mikroukłady VLSI mogą zawierać ponad 100 000 elementów.

Przy szybkim postępie technologii, częstym wprowadzaniu nowych wyrobów, a także ważności oprogramowania i łączności zbliżonej do ważności samego sprzętu, klasyfikacja generacji staje się mniej jasna i mniej znacząca. Można stwierdzić, że główne zmiany spowodowane komercyjnym wykorzystaniem nowych rozwiązań nastąpiły we wczesnych latach siedemdziesiątych, a wyniki tych zmian są wciąż jeszcze eksploatowane. W tym podrozdziale przedstawiamy dwa najważniejsze spośród tych wyników.

Pamięć półprzewodnikowa

Pierwszym zastosowaniem technologii układów scalonych w komputerach było zbudowanie procesora (jednostki sterującej oraz jednostki arytmetyczno-logicznej) z układów scalonych. Stwierdzono jednak, że ta sama technologia może być użyta do budowy pamięci.

W latach pięćdziesiątych i sześćdziesiątych większość pamięci komputerowych była zbudowana z niewielkich pierścieni z materiału ferromagnetycznego, z których każdy miał średnicę około 1/16 cala (ok. 1,6 mm). Pierścienie te były nawlekane na siatki cienkich drutów, które zawieszano na małych ekranach wewnątrz komputera. Po namagnesowaniu w jednym kierunku pierścień (zwany *rdzeniem*) reprezentował jedynkę; namagnesowanie w przeciwnym kierunku reprezentowało zero. Pamięć na rdzeniach magnetycznych była raczej szybka; odczytanie bitu przechowywanego w pamięci zabierało około jednej milionowej części sekundy. Była jednak droga, duża i wykorzystywała odczyt niszczący; prosty akt odczytania rdzenia wymazywał przechowywaną w nim informację. Było więc konieczne zainstalowanie układów regenerujących dane tuż po ich odczytaniu.

Następnie, w roku 1970, firma Fairchild wyprodukowała pierwszą względnie pojemną pamięć półprzewodnikową. Mikroukład ten, o rozmiarze pojedynczego rdzenia, mógł pamiętać 256 bitów. Odczyt był nieniszczący i znacznie szybszy niż w przypadku rdzeni. Odczytanie bitu zabierało zaledwie 70 miliardowych części sekundy. Jednak koszt przypadający na bit był wyższy niż dla rdzeni.

W roku 1974 zaszło wydarzenie brzemienne w skutki: cena przypadająca na bit pamięci półprzewodnikowych spadła poniżej tej ceny pamięci rdzeniowej. W dalszym ciągu następował ciągły i szybki spadek kosztu pamięci, któremu towarzyszył odpowiedni wzrost fizycznej gęstości pamięci. Utorowało to drogę do mniejszych i szybszych maszyn z pamięciami o rozmiarach takich samych, jakie występowały w większych i droższych komputerach jeszcze kilka lat temu.

Rozwój technologii pamięci wraz z rozwojem technologii procesorów przedyskutujemy w dalszej części książki. Zmienił on naturę komputerów w czasie krótszym od dekady. Chociaż wielkie, drogie maszyny pozostały częścią krajobrazu, komputery stały się dostępne użytkownikowi w postaci maszyn biurowych i komputerów osobistych.

Od roku 1970 było jedenaście generacji pamięci półprzewodnikowych: 1 K, 4 K, 16 K, 64 K, 256 K, 1 M, 4 M, 16 M, 64 M, 256 M i obecnie 1 G bitów w jednym mikroukładzie (1 K = 2^{10}, 1 M = 2^{20}, 1 G = 2^{30}). Każda generacja przyniosła 4-krotny wzrost gęstości pamięci w stosunku do poprzedniej generacji, czemu towarzyszyło zmniejszenie kosztu na bit i zmniejszenie czasu dostępu.

Mikroprocesory

W czasie gdy rosła gęstość elementów w mikroukładach pamięciowych, wzrastała również gęstość elementów w mikroukładach procesorowych. W miarę upływu czasu w każdej mikrostrukturze umieszczano coraz więcej elementów, dzięki czemu do budowy pojedynczego procesora była potrzebna coraz mniejsza liczba mikroukładów.

Przełom osiągnięto w roku 1971, kiedy to w firmie Intel opracowano układ 4004. Był to pierwszy mikroukład zawierający *wszystkie* elementy procesora w jednym mikroukładzie; narodził się *mikroprocesor*.

Mikroprocesor 4004 mógł dodawać dwie liczby 4-bitowe, ale mnożyć mógł tylko przez wielokrotne dodawanie. Według dzisiejszych standardów 4004 był beznadziejnie prymitywny, jednak zapoczątkował ciągłą ewolucję możliwości i mocy mikroprocesorów.

Ewolucję tę najłatwiej przeanalizować, rozważając liczbę bitów, którymi operuje procesor w określonym momencie. Nie istnieje jasna miara tej możliwości, jednak być może najlepszą miarą jest szerokość magistrali danych: liczba bitów danych, które mogą być wprowadzane lub wyprowadzane z procesora w ustalonej chwili. Inną miarą jest liczba bitów w akumulatorze lub w zespole rejestrów ogólnego przeznaczenia. Często miary te są zgodne, jednak nie zawsze. Jest na przykład pewna liczba mikroprocesorów, które operują na liczbach 16-bitowych w rejestrach, ale mogą czytać lub zapisywać w tym samym czasie jedynie 8 bitów.

Następnym głównym krokiem w ewolucji mikroprocesorów było wprowadzenie przez firmę Intel w roku 1972 modelu 8008. Był to pierwszy mikroprocesor 8-bitowy, prawie dwa razy bardziej złożony od 4004.

Żaden z tych kroków nie miał jednak tak istotnego wpływu, jak następne ważne wydarzenie: w roku 1972 Intel wprowadził układ 8080. Był to pierwszy mikroprocesor ogólnego przeznaczenia. Podczas gdy 4004 i 8008 zostały zaprojekto-

wane dla szczególnych zastosowań, 8080 zaprojektowano jako jednostkę centralną mikrokomputera ogólnego przeznaczenia. Podobnie jak 8008, 8080 jest mikroprocesorem 8-bitowym. Jednak 8080 jest szybszy, ma bogatszą listę rozkazów i większe możliwości adresowania.

Prawie równocześnie zaczęto opracowywać mikroprocesory 16-bitowe. Jednak dopiero w końcu lat siedemdziesiątych pojawiły się potężne, 16-bitowe mikroprocesory ogólnego przeznaczenia. Jednym z nich był mikroprocesor 8086. Następny krok w tym rozwoju miał miejsce w roku 1981, kiedy to zarówno w Bell Labs, jak i w firmie Hewlett-Packard opracowano jednoukładowy mikroprocesor 32-bitowy. Intel wprowadził swój mikroprocesor 32-bitowy – 80386 – w roku 1985 (tab. 2.6).

Tabela 2.6. Ewolucja mikroprocesorów firmy Intel

(a) Procesory z lat siedemdziesiątych

	4004	8008	8080	8086	8088
Data wprowadzenia na rynek	15.11.1971	1.04.1972	1.04.1974	8.06.1978	1.06.1979
Częstotliwość zegara	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Szerokość magistrali	4 bity	8 bitów	8 bitów	16 bitów	8 bitów
Liczba tranzystorów (mikrony)	2300 (10)	3500	6000 (6)	29 000 (3)	29 000 (3)
Pamięć adresowalna	640 bajtów	16 KB	64 KB	1 MB	1 MB
Pamięć wirtualna	–	–	–	–	–

(b) Procesory z lat osiemdziesiątych

	80286	386TM DX	386TM SX	486TM DX CPU
Data wprowadzenia na rynek	1.02.1982	17.10.1985	16.06.1988	10.04.1989
Częstotliwość zegara	6÷12,5 MHz	16÷33 MHz	16÷33 MHz	25÷50 MHz
Szerokość magistrali	16 bitów	32 bity	16 bitów	32 bity
Liczba tranzystorów (mikrony)	134 000 (1,5)	275 000 (1)	275 000 (1)	1,2 miliona (0,8÷1)
Pamięć adresowalna	16 megabajtów	4 gigabajty	4 gigabajty	4 gigabajty
Pamięć wirtualna	1 gigabajt	64 terabajty	64 terabajty	64 terabajty

(c) Procesory z lat dziewięćdziesiątych

	486TM SX	Pentium	Pentium	Pentium II
Data wprowadzenia na rynek	22.04.1991	22.03.1993	1.11.1995	7.15.1997
Częstotliwość zegara	16÷133 MHz	60÷166 MHz	150÷200 MHz	200÷300 MHz
Szerokość magistrali	32 bity	32 bity	64 bity	64 bity
Liczba tranzystorów (mikrony)	1,185 miliona (1)	3,1 miliona (0,8)	5,5 miliona (0,6)	7,5 miliona (0,35)
Pamięć adresowalna	4 gigabajty	4 gigabajty	64 gigabajty	64 gigabajty
Pamięć wirtualna	64 terabajty	64 terabajty	64 terabajty	64 terabajty

Tabela 2.6 (cd.)

(d) Procesory współczesne

	Pentium III	Pentium 4
Data wprowadzenia na rynek	26.02.1999	11.2000
Częstotliwość zegara	450÷660 MHz	1,3÷1,8 GHz
Szerokość magistrali	64 bity	64 bity
Liczba tranzystorów (mikrony)	9,5 miliona (0,25)	42 miliony
Pamięć adresowalna	64 gigabajty	64 gigabajty
Pamięć wirtualna	64 terabajty	64 terabajty

Źródło: Intel Corp. http://www.intel.com/intel/museum/25anniv/hof/tspecs.htm

2.2. Projektowanie zorientowane na wydajność

Rok po roku koszt systemów komputerowych malał w istotnym stopniu, podczas gdy wydajność i pojemność tych systemów rosły równie szybko. W wielu sklepach możemy wybrać komputer osobisty w cenie poniżej 1000 dolarów, który bije na głowę komputery IBM sprzed 10 lat. We wnętrzu tego komputera osobistego mogą się znajdować setki milionów tranzystorów. Nie możemy kupić 100 milionów czegokolwiek tak tanio. Taka liczba listków papieru toaletowego kosztuje ponad 100 000 dolarów.

Otrzymujemy więc moc komputera praktycznie za darmo, a ta nieustająca rewolucja technologiczna umożliwiła rozwój zastosowań o zdumiewającej złożoności i mocy. Na przykład dzisiejsze biurkowe systemy komputerowe wykorzystujące mikroprocesory umożliwiają m.in.:

- przetwarzanie obrazów;
- rozpoznawanie mowy;
- wideokonferencje;
- opracowania multimedialne;
- dźwiękowe i wizyjne ilustrowanie plików;
- modelowanie symulacyjne.

Komputerowe stacje robocze są obecnie wykorzystywane w niezwykle złożonych opracowaniach technicznych i naukowych, a także w systemach symulacyjnych oraz wspomagają pracę grupową nad obrazami i produktami wideo. Ponadto przedsiębiorcy wykorzystują serwery o rosnącej mocy do obsługi transakcji, przetwarzania baz danych oraz do tworzenia rozbudowanych sieci klient-serwer, które zastąpiły potężne centra komputerowe z minionych lat.

Z perspektywy rozwoju organizacji i architektury komputerów najbardziej fascynujące jest to, że z jednej strony podstawowe bloki konstrukcyjne dzisiejszych „cudów komputerowych" są praktycznie takie same jak w komputerze IAS sprzed

50 lat, podczas gdy z drugiej strony metody „wyciskania ostatnich kropli" wydajności z dostępnych materiałów stały się tak wyrafinowane.

Powyższa obserwacja stała się myślą przewodnią tej książki. Podczas rozpatrywania różnych elementów i zespołów komputera mamy na uwadze dwa cele: po pierwsze – wyjaśnienie fundamentalnej funkcjonalności w każdym rozpatrywanym obszarze, po drugie zaś – zbadanie metod osiągania maksymalnej wydajności. W pozostałej części tego podrozdziału przedstawimy pewne czynniki wzmacniające potrzebę projektowania zorientowanego na wydajność.

Szybkość mikroprocesora

Tym, co daje mikroprocesorom Pentium i PowerPC tak wielką, pobudzającą umysł moc, jest nieugięte dążenie producentów struktur procesorowych do szybkości. Ich ewolucja nadal przebiega zgodnie z przedstawionym powyżej prawem Moore'a. Tak długo, jak prawo to zachowuje swoją moc, wytwórcy mikroukładów mogą wprowadzać nowe generacje mikroukładów – zawierające czterokrotnie więcej tranzystorów – co trzy lata. W przypadku mikroukładów pamięciowych, proces ten doprowadził do 4-krotnego zwiększania pojemności pamięci dynamicznej o swobodnym dostępie (DRAM) – wciąż stanowiącej podstawowe rozwiązanie głównej pamięci komputerów – również co 3 lata. Natomiast w przypadku mikroprocesorów dodanie nowych układów oraz zwiększenie szybkości wynikające ze zmniejszenia odległości między nimi poprawiało wydajność 4-, a nawet 5-krotnie w ciągu każdych 3 lat od czasu wprowadzenia przez firmę Intel w roku 1978 rodziny *X*86.

Jednak sama tylko szybkość mikroprocesora mogłaby pozostać zaledwie potencjalną możliwością, jeśli nie byłby on „zasilany" ciągłym strumieniem zadań do wykonania w postaci rozkazów komputerowych. Wszystko, co staje na drodze temu ciągłemu napływowi rozkazów, obniża moc procesora. Dlatego właśnie, podczas gdy wytwórcy mikroukładów uczyli się, jak wytwarzać struktury o wciąż większej gęstości, projektanci procesorów musieli korzystać z coraz bardziej wymyślnych metod „karmienia" tych potworów. Wśród rozwiązań wbudowanych we współczesne procesory znajdują się:

- **Przewidywanie rozgałęzień** (*branch prediction*). Procesor z wyprzedzeniem przegląda programy i przewiduje, jakie rozgałęzienia lub grupy rozkazów mogą być przetwarzane w dalszej kolejności. Jeśli procesor w większości przypadków zgaduje prawidłowo, to może wstępnie pobierać prawidłowe rozkazy i buforować je, mając w ten sposób ciągle zajęcie. W bardziej wyrafinowanych przykładach zastosowania takiej strategii procesor potrafi przewidywać nie tylko jedno, ale wiele rozgałęzień. Przewidywanie rozgałęzień zwiększa więc ilość pracy dostępnej do wykonania przez procesor.
- **Analiza przepływu danych.** Procesor analizuje, które rozkazy są zależne od wyników innych rozkazów lub od danych w celu stworzenia optymalnego harmonogramu rozkazów. Rozkazy mogą być uszeregowane do wykonania wcześniej, nie-

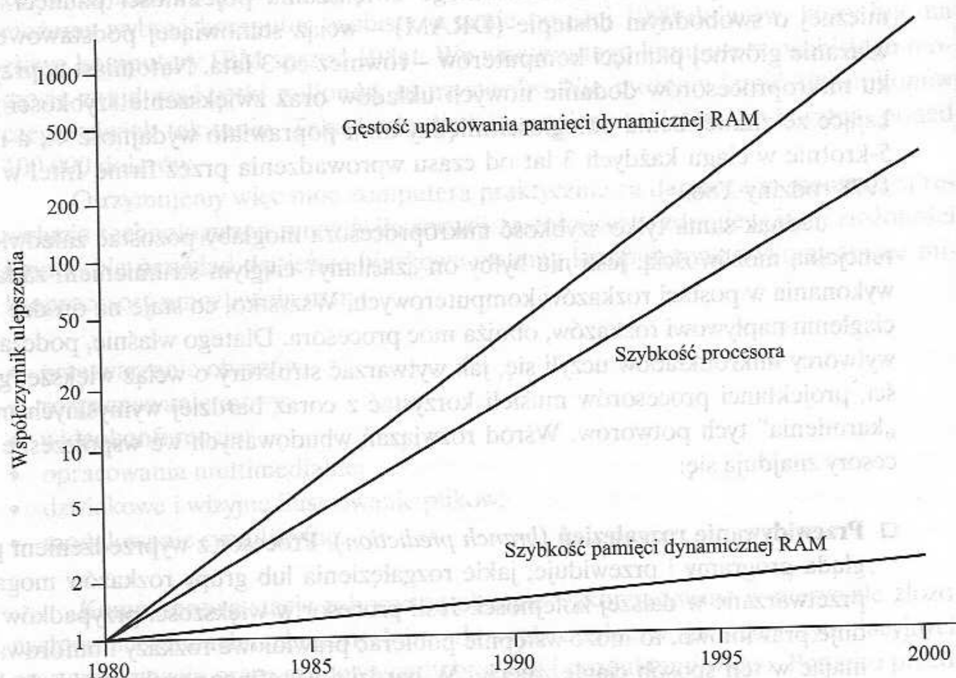
zależnie od oryginalnego polecenia wynikającego z programu. Zapobiega to zbędnym opóźnieniom.

- **Spekulatywne wykonywanie rozkazów.** Wykorzystując przewidywanie skoków oraz analizę przepływu danych, niektóre procesory „spekulatywnie" wykonują rozkazy, zanim pojawią się one w programie, przechowując wyniki w lokacjach tymczasowych. Dzięki temu – wykonując rozkazy, które prawdopodobnie będą potrzebne – procesor może maksymalnie wykorzystywać swoje mechanizmy wykonywania rozkazów.

Te i inne wyrafinowane metody stały się konieczne wobec ogromnej mocy procesora. Umożliwiają one wykorzystanie jego szybkości.

Równoważenie wydajności

Podczas gdy moc procesorów rosła z zawrotną szybkością, rozwój innych krytycznych zespołów komputera był zbyt wolny. Powstała potrzeba szukania równowagi wydajności: dostosowania organizacji i architektury w celu skompensowania niezgodności między możliwościami różnych zespołów.

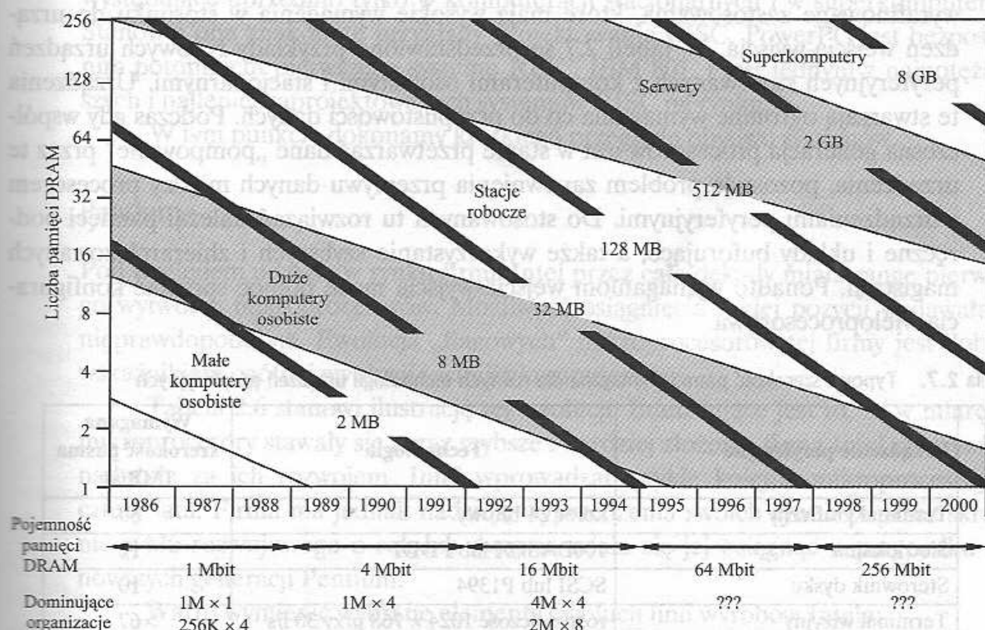


Rysunek 2.10. Ewolucja własności pamięci DRAM i procesorów

Nigdzie indziej problem tego niedopasowania nie jest bardziej krytyczny niż w interfejsie między procesorem a pamięcią główną. Rozważmy historię przedstawioną na rys. 2.10. Podczas gdy szybkość procesora i pojemność pamięci rosły szybko, możliwa do osiągnięcia szybkość przesyłania danych między pamięcią główną

a procesorem stawała się daleko niewystarczająca. Interfejs między procesorem a pamięcią główną jest najbardziej krytycznym elementem całego komputera, ponieważ jest on odpowiedzialny za ciągły przepływ rozkazów i danych między mikroukładami pamięciowymi a procesorem. Jeśli pamięć lub ścieżka danych nie nadążają za potrzebami procesora, pozostaje on w stanie oczekiwania i cenny czas przetwarzania jest tracony.

Wyniki tych tendencji są pokazane na rys. 2.11. Wielkość wymaganej pamięci głównej wzrasta, lecz gęstość pamięci DRAM rośnie jeszcze szybciej. W rezultacie przeciętna liczba mikroukładów DRAM w systemie maleje. Grube czarne linie na rysunku pokazują, że przy ustalonej pojemności pamięci liczba wymaganych układów DRAM maleje. To jednak ma wpływ na szybkość przesyłania, ponieważ mniejsza liczba układów DRAM oznacza mniej możliwości równoległego przesyłania danych. Zacieniowane pasma pokazują, że dla danego typu systemu rozmiar pamięci głównej rósł powoli, podczas gdy liczba układów DRAM malała.



Rysunek 2.11. Tendencje rozwoju zastosowań pamięci DRAM [PRZY94]

Są sposoby, z których architekt systemów może korzystać w celu rozwiązania tego problemu. Wszystkie znajdują odbicie we współczesnych projektach komputerów. Oto przykłady:

- Zwiększanie liczby bitów, które są jednocześnie wyprowadzane, przez projektowanie raczej „szerokich" niż „głębokich" układów DRAM, a także stosowanie szerokich magistrali danych.

- Stosowanie bardziej efektywnych interfejsów pamięci DRAM przez umieszczanie pamięci podręcznych lub innych układów buforowych w strukturach DRAM.
- Redukowanie częstości sięgania do pamięci przez wprowadzanie coraz bardziej złożonych i efektywnych struktur pamięci podręcznych między procesorem a pamięcią główną. Do tych rozwiązań należy umieszczanie jednej lub wielu pamięci podręcznych w mikroukładzie procesora, jak również umieszczanie odrębnej pamięci podręcznej w pobliżu struktury procesora.
- Zwiększanie szerokości pasma przenoszenia między procesorami a pamięcią przez wykorzystanie magistrali o dużej szybkości oraz wprowadzanie hierarchicznego systemu magistrali w celu buforowania i strukturalizowania przepływu danych.

Innym obszarem skupiającym uwagę projektantów są urządzenia wejścia-wyjścia. W miarę wzrostu szybkości i możliwości komputerów, powstają bardziej wyrafinowane zastosowania, które mają wysokie wymagania w stosunku do urządzeń wejścia-wyjścia. W tabeli 2.7 są przedstawione przykłady typowych urządzeń peryferyjnych stosowanych z komputerami osobistymi i stacjonarnymi. Urządzenia te stwarzają ogromne wymagania co do przepustowości danych. Podczas gdy współczesna generacja procesorów jest w stanie przetwarzać dane „pompowane" przez te urządzenia, pozostaje problem zapewnienia przepływu danych między procesorem a urządzeniami peryferyjnymi. Do stosowanych tu rozwiązań należą: pamięci podręczne i układy buforujące, a także wykorzystanie szybszych i zhierarchizowanych magistrali. Ponadto wymaganiom wejścia-wyjścia może pomóc sprostać konfiguracja wieloprocesorowa.

Tabela 2.7. Typowa szerokość pasma wymagana dla różnych technologii urządzeń peryferyjnych

Urządzenie peryferyjne	Technologia	Wymagana szerokość pasma [MB/s]
Terminal graficzny	kolor 24-bitowy	30
Sieć lokalna	100BASEX lub FDDI	12
Sterownik dysku	SCSI lub P1394	10
Terminal wizyjny	rozdzielczość 1024 × 768 przy 30 f/s	>67
Inne urządzenia wejścia-wyjścia	różne	>5

Kluczową sprawą jest w tym wszystkim równowaga. Projektanci stale dążą do zrównoważenia przepustowości oraz wymagań odnoszących się do przetwarzania między procesorem, pamięcią główną, urządzeniami wejścia-wyjścia i strukturami połączeń. Projekty muszą być wciąż analizowane od nowa wobec dwóch stale ewoluujących czynników:

- Szybkości zmian wydajności procesorów, magistrali, pamięci i urządzeń peryferyjnych różnią się znacznie.

- Nowe zastosowania i nowe urządzenia peryferyjne ciągle zmieniają naturę wymagań w stosunku do systemu, wyrażoną poprzez typowy profil rozkazów oraz wzór dostępu do danych.

Tak więc projektowanie komputerów jest sztuką stale ewoluującą. Książka ta ma na celu przedstawienie podstaw tej sztuki oraz jej aktualnego stanu.

2.3. Ewolucja Pentium i PowerPC

W książce tej zostało wykorzystanych wiele konkretnych przykładów projektów komputerów w celu zilustrowania koncepcji oraz wyjaśnienia niezbędnych kompromisów. W większości są to przykłady pochodzące z dwóch rodzin komputerów: Pentium firmy Intel i PowerPC. Pentium jest rezultatem trwającej już dekady pracy nad komputerami o złożonej liście rozkazów (CISC). Zawarto w nich rozwiązania występujące uprzednio tylko w komputerach stacjonarnych i w superkomputerach. Stanowią one znakomite przykłady projektowania CISC. PowerPC jest bezpośrednim potomkiem pierwszego systemu RISC, IBM 801, i jest jednym z najpotężniejszych i najlepiej zaprojektowanych systemów RISC na rynku.

W tym punkcie dokonamy krótkiego przeglądu obydwu systemów.

Pentium

Pod względem udziału w rynku firma Intel przez całe dekady miała rangę pierwszego wytwórcy mikroprocesorów. Możliwość osiągnięcia takiej pozycji wydawała się nieprawdopodobna. Ewolucja „flagowych" mikroprocesorów tej firmy jest dobrym wskaźnikiem ogólnej ewolucji techniki komputerowej.

Tabela 2.6 stanowi ilustrację tej ewolucji. Interesujące jest to, że w miarę jak mikroprocesory stawały się coraz szybsze i bardziej złożone, firma Intel rzeczywiście nadążała za ich rozwojem. Intel wprowadzał zwykle kolejne mikroprocesory co cztery lata. Firma ma jednak nadzieję wyprzedzenia swoich rywali poprzez skrócenie cyklu rozwojowego o rok lub dwa, co udało się jej osiągnąć w przypadku najnowszych generacji Pentium.

Warto wymienić wybrane elementy ewolucji linii wyrobów Intela:

- **8080.** Pierwszy na świecie mikroprocesor ogólnego przeznaczenia. Było to urządzenie 8-bitowe, z 8-bitową ścieżką danych do pamięci. Mikroprocesor 8080 został użyty w pierwszym komputerze osobistym „Altair".
- **8086.** Daleko potężniejsze, 16-bitowe urządzenie. Poza szerszą ścieżką danych i większymi rejestrami, mikroprocesor 8086 zawiera podręczną pamięć rozkazów, która wstępnie pobiera kilka rozkazów przed ich wykonaniem. Odmiana tego procesora, 8088, została użyta w pierwszym komputerze osobistym firmy IBM, umacniając sukces Intela.
- **80286.** Stanowi on rozszerzenie mikroprocesora 8086, umożliwiając adresowanie pamięci 16 MB zamiast tylko 1 MB.

- **80386.** Pierwsze 32-bitowe urządzenie Intela, jednocześnie znaczne odnowienie linii mikroprocesorów. Mając architekturę 32-bitową, mikroprocesor 80386 rywalizował pod względem złożoności i mocy z minikomputerami i komputerami stacjonarnymi wprowadzonymi zaledwie kilka lat wcześniej. Był to pierwszy procesor firmy Intel wspierający wielozadaniowość, co oznacza, że mógł wykonać jednocześnie wiele programów.
- **80486.** W tym mikroprocesorze wprowadzono znacznie bardziej złożoną i potężną technikę pamięci podręcznej, a także wyrafinowane potoki rozkazów. Procesor 80486 zawierał również wbudowany koprocesor arytmetyczny, odciążający procesor główny od wykonywania złożonych operacji matematycznych.
- **Pentium.** Wraz z Pentium Intel wprowadził do użytku techniki superskalarne, co umożliwiło równoległe wykonywanie wielu rozkazów.
- **Pentium Pro.** Pentium Pro był świadectwem dalszego – zapoczątkowanego wraz z Pentium – angażowania się w organizację superskalarną, przy czym w znacznie szerszym zakresie zastosowano przemianowywanie rejestrów, przewidywanie rozgałęzień, analizę przepływu danych i spekulatywne wykonywanie rozkazów.
- **Pentium II.** W Pentium II zastosowano technologię MMX firmy Intel, zaprojektowaną pod kątem efektywnego przetwarzania danych wideo, audio i graficznych.
- **Pentium III.** W Pentium III wprowadzono dodatkowe rozkazy zmiennopozycyjne w celu wspierania oprogramowania przeznaczonego do przetwarzania grafiki trójwymiarowej.
- **Pentium 4.** W Pentium 4 zastosowano dodatkowe rozkazy zmiennopozycyjne i inne udoskonalenia związane z przetwarzaniem danych multimedialnych[2].
- **Itanium.** W tej nowej generacji procesorów firmy Intel zastosowano organizację 64-bitową i architekturę IA-64, która zostanie przedstawiona szczegółowo w rozdziale 15.

PowerPC

W roku 1975 w firmie IBM opracowano minikomputer 801, w którym po raz pierwszy wprowadzono wiele koncepcji architektury stosowanych w systemach RISC. Mikrokomputer 801 wraz z procesorem RISC I z Berkeley zapoczątkował rozwój tych systemów. Jednakże 801 był tylko prototypem przeznaczonym do zademonstrowania koncepcji projektowych. Sukces mikrokomputera 801 skłonił IBM do opracowania komercjalnego produktu RISC, pod nazwą RTPC. W maszynie RTPC wprowadzonej na rynek w roku 1986, koncepcje architektoniczne mikrokomputera 801 zostały zastosowane w rzeczywistym produkcie. Maszyna ta nie była sukcesem komercyjnym i miała wielu rywali o porównywalnej i większej wydajności. W roku 1990 IBM wprowadził trzeci system, w którym wykorzystano lekcję 801 i RT PC. RISC System/6000

[2] Wraz z Pentium 4 firma Intel zmieniła sposób numerowania kolejnych procesorów, zastępując cyfry rzymskie arabskimi.

był zbliżoną do RISC maszyną superskalarną sprzedawaną jako urządzenie stacjonarne o wysokiej wydajności. Wkrótce po jego wprowadzeniu IBM zaczął określać jego architekturę mianem POWER.

W dalszym ciągu IBM nawiązał współpracę z firmą Motorola, w której opracowano szereg mikroprocesorów 68000, oraz z firmą Apple, która wykorzystywała mikroukłady Motoroli w swoich komputerach Macintosh. Wynikiem współpracy była seria maszyn wykorzystujących architekturę PowerPC. Architektura ta wywodzi się z architektury POWER. Dokonano zmian, uzupełniając brakujące własności, oraz umożliwiając bardziej efektywne wdrożenie przez wyeliminowanie niektórych rozkazów. Złagodzono pewne wymagania w celu uniknięcia kłopotliwych problemów w niektórych szczególnych przypadkach. Wynikająca stąd architektura PowerPC jest systemem superskalarnym RISC. PowerPC jest używany w milionach komputerów Macintosh firmy Apple i w licznych zastosowaniach jako procesor wbudowany. Przykładem takich zastosowań jest wytwarzana przez IBM rodzina mikroukładów służących do zarządzania sieciami, które mogą być wbudowywane do sprzętu sieciowego w celu umożliwienia zarządzania użytkownikom platform mających składniki pochodzące od różnych dostawców.

Tabela 2.8. Zestawienie właściwości procesorów PowerPC

	601	603/603e	604/604e	740/750 (G3)	G4
Rok wprowadzenia na rynek	1993	1994	1994	1997	1999
Częstotliwość zegara (MHz)	50÷120	100÷300	166÷350	200÷366	500
Pamięć podręczna L1	–	16 KB rozkazy 16 KB dane	32 KB rozkazy 32 KB dane	32 KB rozkazy 32 KB dane	32 KB rozkazy 32 KB dane
Pomocnicza pamięć podręczna L2	–	–	–	256 KB÷1 MB	256 KB÷1 MB
Liczba tranzystorów (10^6)	2,8	1,6÷2,6	3,6÷5,1	6,35	–

Dotychczas opracowano cztery urządzenia należące do rodziny PowerPC (tabela 2.8):

- **601.** Mikroprocesor 601 wyprodukowano, aby wprowadzić na rynek architekturę PowerPC tak szybko, jak było to możliwe. Jest on maszyną 32-bitową.
- **603.** Mikroprocesor przeznaczony dla komputerów biurkowych o stosunkowo małej wydajności i dla przenośnych. Jest to również maszyna 32-bitowa, porównywalna pod względem wydajności z 601, jednak kosztuje mniej, a wdrożenie jej jest bardziej efektywne.
- **604.** Mikroprocesor przeznaczony dla komputerów biurkowych oraz serwerów o stosunkowo małej wydajności. Jest to również maszyna 32-bitowa, jednak wy-

korzystano w niej bardziej zaawansowane rozwiązania superskalarne w celu zwiększenia wydajności.

- **620.** Mikroprocesor dla serwerów o większej wydajności. Pierwsze urządzenie z rodziny PowerPC, w którym wykorzystano w pełni 64-bitową architekturę, łącznie z 64-bitowymi rejestrami i ścieżkami danych.
- **740/750.** Znany również jako procesor G3. W głównym mikroukładzie procesora zintegrowano dwa poziomy pamięci podręcznej, uzyskując dzięki temu znaczną poprawę wydajności w porównaniu z porównywalnymi rozwiązaniami z oddzielnymi pamięciami podręcznymi.
- **G4.** W tym procesorze zwiększono zakres przetwarzania równoległego i wewnętrzną szybkość mikroukładu procesorowego.

2.4. Polecana literatura i witryny WWW

Opis serii IBM 7000 można znaleźć w [BELL71a]. Wyczerpujący opis IBM 360 znajduje się w [SIEW82], a PDP-8 i innych maszyn DEC w [BELL78a]. Te trzy książki zawierają również liczne, szczegółowe przykłady dotyczące innych komputerów i prezentujące historię komputerów we wczesnych latach osiemdziesiątych. Nowszą książką zawierającą doskonały zbiór opisów komputerów o znaczeniu historycznym jest [BLAA97]. Historia mikroprocesorów jest zawarta w [BETK97].

Jednym z najlepszych opisów Pentium jest [SHAN98]. Dobra jest również dokumentacja samego Intela [INTE01]. Praca [BREY00] stanowi dobry przegląd linii mikroprocesorów Intela, z akcentem na maszyny 32-bitowe.

W [IBM94] wyczerpująco przedstawiono architekturę PowerPC. W pracy [SHAN95] podano podobny zakres zagadnień. W [WEIS94] znajduje się opis zarówno architektury POWER, jak i PowerPC.

Interesująca analiza prawa Moore'a i jego konsekwencji znajduje się w [HUTC96], [SCHA97] i [BOHR98].

BELL71a Bell C., Newell A.: *Computer Structures: Readings and Examples*. New York, McGraw-Hill, 1971.

BELL78a Bell C., Mudge J., McNamara J.: *Computer Engineering: A DEC View of Hardware Systems Design*. Bedford, Digital Press, 1978.

BETK97 Betker M., Fernando J., Whalen S.: The History of Microprocessor. *Bell Labs Technical Journal*, Autumn 1997.

BLAA97 Blaauw G., Brooks F.: *Computer Architecture: Concepts and Evolution*. Reading, Addison-Wesley, 1997.

BOHR98 Bohr M.: Silicon Trends and Limits for Advanced Microprocessors. *Communications of the ACM*, March 1998.

BREY00 Brey B.: *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro and Pentium II Processors*. Upper Saddle River, Prentice Hall, 2000.

HUTC96 Hutcheson G., Hutcheson J.: Technology and Economics in the Semiconductor Industry. *Scientific American*, January 1996.

IBM94 International Business Machines, Inc.: *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, Morgan-Kaufmann, 1994.

INTE01 Intel Corp.: *IA-32 Intel Architecture Software Developer's Manual* (2 volumes). Document 245470 and 245471. Aurora, CO, 2000.

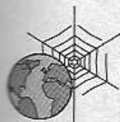
SCHA97 Schaller R.: Moore's Law: Past, Present, and Future. *IEEE Spectrum*, June 1997.

SHAN95. Shanley T.: *PowerPC System Architecture*. Reading, Addison-Wesley, 1998.

SHAN98 Shanley T.: *Pentium Pro and Pentium II System Architecture*. Reading, Addison-Wesley, 1998.

SIEW82 Siewiorek D., Bell C., Newell A.: *Computer Structures: Principles and Examples*. New York, McGraw-Hill, 1982.

WEIS94 Weiss S., Smith J.: *POWER and PowerPC*. San Francisco, Morgan Kaufmann, 1994.



Polecane witryny WWW:

- **Intel Developer's Page.** Strona WWW firmy Intel przeznaczona dla projektantów oprogramowania; punkt wyjścia do poszukiwania informacji na temat procesorów Pentium. Obejmuje również *Intel Technology Journal*.
- **PowerPC.** Dwie witryny WWW poświęcone PowerPC, jedna prowadzona przez IBM, druga zaś przez firmę Motorola.
- **Top500 Supercomputer Site.** Zawiera krótki opis architektury i organizacji współczesnych superkomputerów oraz ich porównanie.
- **Charles Babbage Institute.** Zawiera łącza do witryn WWW poświęconych historii komputerów.

2.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

akumulator (AC) – *accumulator*
cykl pobierania – *fetch cycle*
cykl rozkazu – *instruction cycle*
cykl wykonywania – *execute cycle*
jednostka arytmetyczno-logiczna (ALU) – *arithmetic and logic unit*
jednostka sterowania programem – *program control unit*
kanał danych – *data channel*
kod rozkazu – *opcode*
kompatybilność wstępująca – *upward compatibility*
komputer z przechowywanym programem – *stored program computer*
licznik rozkazów – *instruction register*
lista rozkazów – *instruction set*
maszyna von Neumanna – *von Neumann machine*
mikroprocesor – *microprocessor*
mikroukład – *chip*
multiplekser – *multiplexer*
pamięć główna – *main memory*
płytka – *wafer*
rejestr adresowy pamięci (MAR) – *memory address register*
rejestr buforowy pamięci (MBR) – *memory buffer register*
rejestr buforowy rozkazów (IBR) – *instruction buffer register*
rejestr rozkazów (IR) – *instruction register*
słowo – *word*
układ scalony (IC) – *integrated circuit*
wejście-wyjście – *input/output (I/O)*
wytwórca sprzętu oryginalnego (OEM) – *original equipment manufacturer*

Pytania kontrolne

2.1. Co to jest komputer z przechowywanym programem?

2.2. Jakie są cztery główne zespoły komputera o przeznaczeniu ogólnym?

2.3. Jakie są trzy podstawowe składniki systemu komputerowego na poziomie układów scalonych?

2.4. Objaśnij prawo Moore'a.

2.5. Wymień i objaśnij podstawowe właściwości rodziny komputerów.

2.6. Co jest podstawową cechą wyróżniającą mikroprocesory?

Problemy do rozwiązania

2.1. Niech A = A(1), A(2), ..., A(1000) oraz B = B(1), B(2), ..., B(1000) będą dwoma wektorami (tablicami jednowymiarowymi) składającymi się z tysiąca liczb każdy, które mają być dodane w celu utworzenia tablicy C takiej, że C(I) = A(I) + B(I) dla I = 1, 2, ..., 1000. Stosując listę rozkazów IAS, napisz program do rozwiązania tego problemu.

2.2. W należących do serii IBM 360 modelach 65 i 75 adresy są gromadzone w dwóch oddzielnych jednostkach pamięciowych (np. wszystkie słowa numerowane parzyście w jednej jednostce, a wszystkie numerowane nieparzyście – w drugiej). Jaki może być cel takiego rozwiązania?

Część druga
System komputerowy

Problemy przedstawione w części drugiej

System komputerowy składa się z procesora, pamięci, urządzeń wejścia-wyjścia oraz z połączeń między tymi głównymi zespołami. Z wyjątkiem procesora, który jest na tyle złożony, że poświęcono mu oddzielną część trzecią, zespoły te zostaną przeanalizowane szczegółowo w części drugiej.

Tematyka poszczególnych rozdziałów części drugiej

Rozdział 3. Ogólny obraz działania komputera i jego połączeń wewnętrznych

Na najwyższym poziomie organizacji komputer składa się z procesora, pamięci i urządzeń wejścia-wyjścia. Działanie takiego systemu obejmuje wymianę danych i sygnałów sterujących między tymi zespołami. Aby taka wymiana mogła zachodzić, zespoły te muszą być ze sobą połączone. Rozdział 3 rozpoczyna się od krótkiego omówienia zespołów komputera i odnoszących się do nich wymagań w kategoriach wejścia-wyjścia. Następnie są przedstawione podstawowe problemy projektowania wzajemnych połączeń, a zwłaszcza konieczność obsługi przerwań. Główna część tego rozdziału jest poświęcona analizie najczęściej stosowanego rozwiązania wzajemnych połączeń: strukturze magistrali.

Rozdział 4. Pamięć podręczna

Pamięci stosowane w komputerze różnią się pod względem rodzaju, technologii, organizacji, wydajności i kosztu. W typowym systemie komputerowym występuje cała hierarchia podsystemów pamięciowych; niektóre z nich są wewnętrzne (bezpośrednio dostępne dla procesora), niektóre zaś zewnętrzne (dostępne dla procesora poprzez moduł wejścia-wyjścia). Rozdział 4 rozpoczyna się od przeglądu tej hierarchii. Następnie jest szczegółowo przedstawione projektowanie pamięci podręcznej, w tym oddzielnych pamięci kodu i danych oraz pamięci dwupoziomowych.

Rozdział 5. Pamięć wewnętrzna

Projektowanie systemu pamięci głównej to niekończąca się batalia między trzema rywalizującymi ze sobą wymaganiami projektowymi: dużą pojemnością, krótkim czasem dostępu i nis-

kim kosztem. W miarę rozwijania się technologii pamięci każda z tych cech ulega zmianie, więc przy każdej nowej implementacji decyzje projektowe dotyczące organizacji pamięci głównej muszą być rozważone od nowa. W rozdziale 5 skoncentrujemy się na problemach projektowania pamięci wewnętrznej. Po pierwsze, przeanalizujemy naturę i organizację półprzewodnikowej pamięci głównej. Następnie przedstawimy najnowsze rozwiązania organizacyjne pamięci DRAM.

Rozdział 6. Pamięć zewnętrzna

Aby było możliwe uzyskanie rzeczywiście dużej zdolności trwałego przechowywania danych, potrzebna jest pamięć zewnętrzna. Najczęściej stosowanym rodzajem pamięci zewnętrznej są dyski magnetyczne, którym poświęcimy większość rozdziału 6. Po pierwsze, przedstawimy technologię dysków magnetycznych i rozważania projektowe. Następnie zapoznamy się z organizacją RAID mającą na celu zwiększenie wydajności pamięci dyskowej. W rozdziale 6 również przeanalizujemy pamięci optyczne i taśmowe.

Rozdział 7. Wejście-wyjście

Moduły wejścia-wyjścia są połączone z procesorem i z pamięcią główną, a każdy z nich steruje jednym lub wieloma urządzeniami zewnętrznymi. Rozdział 7 poświęcimy różnym aspektom organizacji wejścia-wyjścia. Jest to dziedzina złożona i zwykle słabiej rozumiana niż pozostałe obszary projektowania systemu komputerowego, jeśli chodzi o osiąganie wymaganej wydajności. W rozdziale 7 przeanalizujemy mechanizmy wzajemnego oddziaływania między modułami wejścia-wyjścia a resztą systemu komputerowego; zostaną uwzględnione techniki programowanego wejścia-wyjścia, przerwań i bezpośredniego dostępu do pamięci (DMA). Przedstawimy również interfejs między modułem wejścia-wyjścia a urządzeniami zewnętrznymi.

Rozdział 8. Wspieranie systemu operacyjnego

Szczegółowa analiza systemów operacyjnych (OS) wykracza poza ramy tej książki. Ważne jest jednak zrozumienie podstawowych funkcji systemu operacyjnego oraz tego, w jaki sposób wykorzystuje on sprzęt w celu zapewnienia pożądanej wydajności. W rozdziale 8 opiszemy podstawowe zasady systemów operacyjnych oraz specyficzne właściwości projektowe sprzętu komputerowego obliczone na zapewnienie obsługi systemu operacyjnego. Rozdział ten rozpoczniemy od krótkiej historii, która posłuży zidentyfikowaniu głównych rodzajów systemów operacyjnych i uzasadnieniu ich używania. Następnie objaśnimy wieloprogramowanie poprzez przeanalizowanie funkcji szeregowania długo- i krótkookresowego. Na zakończenie rozważymy zarządzanie pamięcią z uwzględnieniem segmentowania, stronicowania i pamięci wirtualnej.

Rozdział 3

Ogólny obraz działania komputera i jego połączeń wewnętrznych

3.1. Zespoły komputera
3.2. Działanie komputera
Pobieranie i wykonywanie rozkazu
Przerwania
Działanie modułów wejścia-wyjścia
3.3. Struktura połączeń wewnętrznych
3.4. Połączenia magistralowe
Struktura magistrali
Hierarchiczne struktury wielomagistralowe
Elementy projektowania magistrali
3.5. Magistrala PCI
Struktura magistrali
Rozkazy PCI
Przesyłanie danych
Arbitraż
3.6. Polecana literatura i witryny WWW
3.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania
Dodatek 3A. Wykresy przebiegów czasowych

PODSTAWOWE SPOSTRZEŻENIA

- Cykl rozkazu składa się z pobierania rozkazu, po którym następuje ewentualne pobieranie argumentów (zera lub wielu), zapisywanie ich oraz sprawdzanie przerwania (jeśli przerwania są dozwolone).
- Podstawowe zespoły systemu komputerowego (procesor, pamięć, moduły wejścia-wyjścia) muszą być wzajemnie połączone, aby była możliwa wymiana danych i sygnałów sterujących. Najpopularniejszym rozwiązaniem takiego połączenia jest zastosowanie wspólnej magistrali systemowej składającej się z wielu linii. We współczesnych systemach w celu zwiększenia wydajności stosuje się zwykle magistrale hierarchiczne.
- Do podstawowych problemów projektowych magistrali należą: arbitraż (scentralizowany lub rozproszony sposób udzielania zezwoleń na przesyłanie sygnałów liniami magistralowymi); sposób sterowania przebiegami czasowymi (synchronizacja sygnałów przesyłanych magistralą za pomocą centralnego zegara lub ich przesyłanie asynchroniczne na podstawie ostatniej transmisji) oraz szerokość (liczba linii adresowych i linii danych).

Na najwyższym poziomie organizacji komputer składa się z procesora, pamięci i urządzeń wejścia-wyjścia, przy czym każdy z tych modułów może występować pojedynczo lub w większej ilości. Zespoły te są połączone w sposób umożliwiający realizowanie podstawowej funkcji komputera, jaką jest wykonywanie programów. Wobec tego – pozostając na tym najwyższym poziomie – możemy opisać system komputerowy przez: (1) przedstawienie zewnętrznego zachowania każdego modułu, to znaczy danych i sygnałów kontrolnych, które wymienia on z innymi modułami, oraz (2) podanie struktury połączeń i sterowania wymaganego do zarządzania tą strukturą.

Obraz struktury i funkcji widziany na najwyższym poziomie organizacji jest ważny, ponieważ ułatwia zrozumienie natury komputera. Równie ważne jest jego wykorzystanie dla zrozumienia wciąż komplikujących się zagadnień oceny wydajności. Uchwycenie struktury i funkcji na tym poziomie pozwala zauważyć wąskie gardła sytemu, rozwiązania alternatywne, zasięg uszkodzeń systemu wywoływanych przez uszkodzenie zespołu oraz możliwość zwiększenia wydajności. W wielu przypadkach żądaniu większej mocy i odporności na uszkodzenia systemu można sprostać raczej przez zmiany projektowe, niż tylko przez zwiększanie szybkości i niezawodności poszczególnych zespołów.

W tym rozdziale skupimy się na podstawowych strukturach wykorzystywanych do łączenia zespołów komputera. Rozpoczniemy od krótkiej analizy podstawowych zespołów i ich wymagań interfejsowych. Następnie dokonamy przeglądu funkcjonalnego. Jesteśmy więc gotowi do przeanalizowania problemu wykorzystywania magistrali do łączenia zespołów komputera.

3.1. Zespoły komputera

Jak wykazaliśmy w rozdz. 2, w praktycznie wszystkich współczesnych projektach komputerów wykorzystuje się koncepcje opracowane przez Johna von Neumanna z Institute for Advanced Studies w Princeton. Projekty tego typu są określane jako *architektura von Neumanna*; są w nich wykorzystywane trzy kluczowe koncepcje:

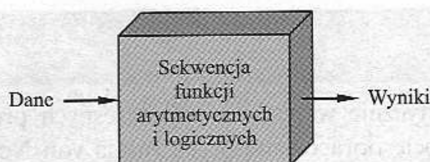
- Dane i rozkazy są przechowywane w tej samej pamięci umożliwiającej zapis i odczyt.
- Zawartość tej pamięci może być adresowana przez wskazanie miejsca, bez względu na rodzaj zawartych tam danych.
- Wykonywanie rozkazów następuje w sposób szeregowy (z wyjątkiem określonych, szczególnych przypadków), rozkaz po rozkazie.

Uwarunkowania leżące u podstaw tych koncepcji przedyskutowaliśmy w rozdz. 1, jednak warto je tutaj podsumować. Istnieje pewien niewielki zestaw podstawowych elementów logicznych, które mogą być łączone na różne sposoby w celu przechowywania danych binarnych oraz wykonywania operacji arytmetycznych i logicznych na tych danych. Jeśli istnieje określony rodzaj obliczenia, które ma być przeprowadzone, to można zbudować specyficzną dla tego obliczenia konfigurację elementów logicznych. Możemy widzieć proces łączenia tych elementów w żądaną konfigurację jako formę programowania. Wynikający stąd „program" ma postać sprzętu i jest nazywany *programem sprzętowym* (*hardwired program*).

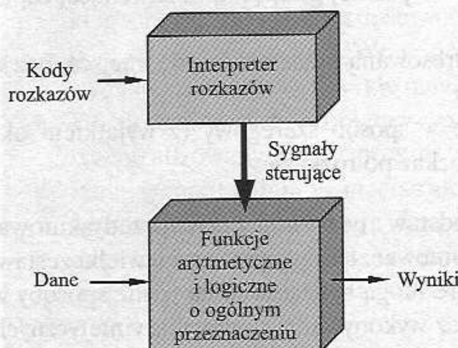
Rozpatrzmy tę alternatywę. Załóżmy, że budujemy zbiór funkcji arytmetycznych i logicznych o ogólnym przeznaczeniu. Urządzenie będzie realizowało różne operacje na danych, zależnie od doprowadzonych sygnałów sterujących. W oryginalnym przypadku urządzenia zaprojektowanego do wykonywania konkretnego zadania system przyjmuje dane i dostarcza wyniki (rys. 3.1a). Natomiast w przypadku urządzenia o ogólnym przeznaczeniu system przyjmuje dane i sygnały sterujące, po czym dostarcza wyniki. Zamiast więc przebudowywać urządzenie dla każdego nowego programu, programista musi tylko dostarczyć nowy zestaw sygnałów sterujących.

Jak mogą być dostarczane sygnały sterujące? Odpowiedź jest prosta, lecz zawiera pewne subtelności. Cały program jest szeregiem (sekwencją) kroków. W każdym z tych kroków jest wykonywana na danych pewna operacja arytmetyczna lub logiczna. Dla każdego kroku jest wymagany nowy zestaw sygnałów sterujących. Spróbujmy teraz przypisać unikatowy kod każdemu z możliwych zestawów sygnałów sterujących i dodajmy do urządzenia o ogólnym przeznaczeniu segment, który przyjmuje kod i generuje sygnały sterujące (rys. 3.1b).

Programowanie jest teraz znacznie łatwiejsze. Zamiast zmieniać połączenia w sprzęcie, musimy jedynie doprowadzić nową sekwencję kodów. Każdy kod jest w rezultacie rozkazem, odpowiednia zaś część urządzenia interpretuje każdy rozkaz i generuje sygnały sterujące. W celu odróżnienia tej nowej metody programowania sekwencja kodów lub rozkazów jest nazywana *oprogramowaniem* (*software*).



(a) Rozwiązanie sprzętowe



(b) Rozwiązania programowe

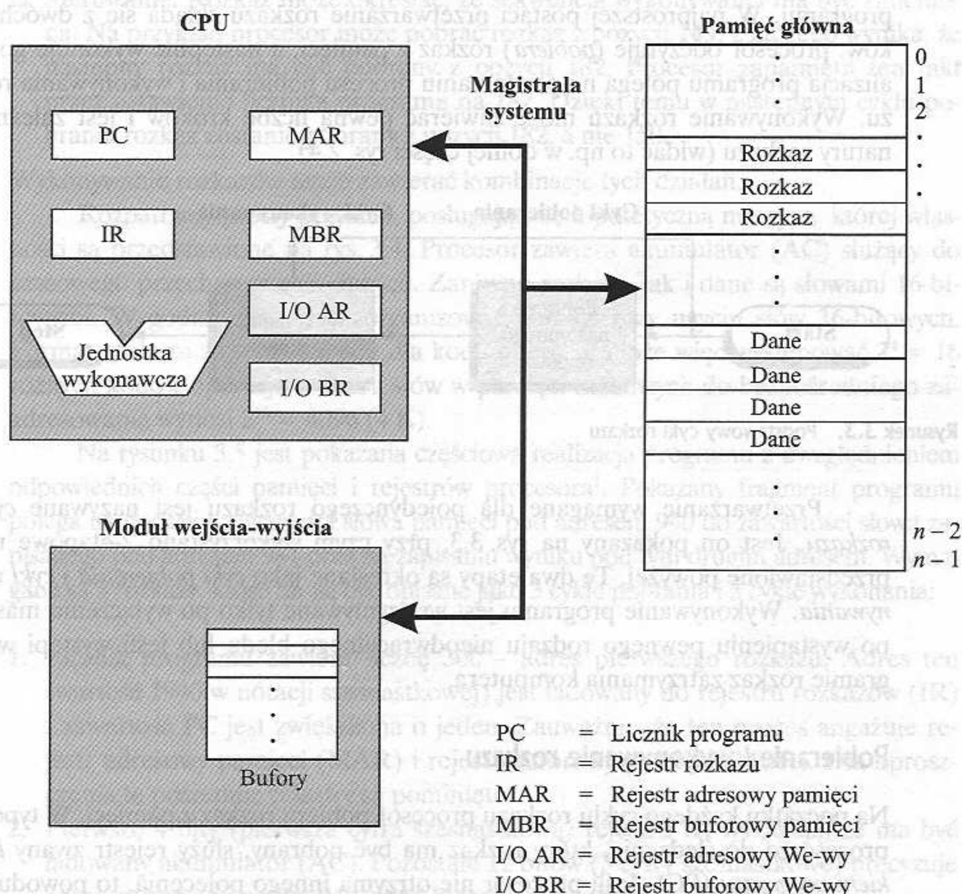
Rysunek 3.1. Rozwiązania sprzętowe i programowe

Na rysunku 3.1b są pokazane dwa główne składniki systemu: moduł interpretujący rozkazy oraz moduł realizujący funkcje arytmetyczne i logiczne. Oba razem tworzą *jednostkę centralną*. Aby uzyskać działający komputer, potrzeba jeszcze paru innych zespołów. Do systemu muszą być wprowadzane dane i rozkazy. Do tego celu potrzebujemy pewnego rodzaju modułu wejściowego. Zawiera on podstawowe podzespoły, które przyjmują dane i rozkazy w pewnej formie, po czym dokonują ich konwersji na wewnętrzną postać sygnałów używaną w systemie. Potrzebne jest też urządzenie prezentujące wyniki, będące modułem wyjściowym. Razem są one określane jako *moduły wejścia-wyjścia*.

Potrzebny jest jeszcze jeden zespół. Urządzenie wejściowe doprowadza dane i rozkazy sekwencyjnie. Jednak program nie zawsze jest realizowany sekwencyjnie; może zawierać skoki (np. rozkaz skoku IAS). Podobnie operacje na danych mogą wymagać dostępu do więcej niż jednego elementu w tym samym czasie, w z góry określonej sekwencji. Musi więc istnieć miejsce do czasowego przechowywania zarówno rozkazów, jak i danych. Odpowiedni moduł nazywa się *pamięcią* lub *pamięcią główną* w odróżnieniu od pamięci zewnętrznej lub pamięci występujących w urządzeniach peryferyjnych. Von Neumann zauważył, że ta sama pamięć może służyć zarówno do przechowywania rozkazów, jak i danych.

Na rysunku 3.2 są pokazane zespoły komputera na najwyższym poziomie organizacji oraz oddziaływanie między nimi. Procesor wymienia dane z pamięcią. Do tego celu są wykorzystywane dwa rejestry wewnętrzne (w stosunku do procesora):

rejestr adresowy pamięci (MAR), określający adres w pamięci następnego zapisu lub odczytu, oraz rejestr buforowy pamięci (MBR) zawierający dane, które mają być zapisane w pamięci lub dane odczytane z pamięci. Podobnie rejestr adresowy wejścia-wyjścia (I/OAR) określa konkretne urządzenie wejścia-wyjścia. Rejestr buforowy wejścia-wyjścia jest wykorzystywany do wymiany danych między modułem wejścia-wyjścia a jednostką centralną.



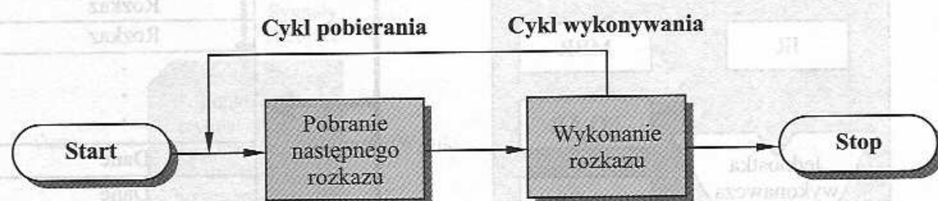
Rysunek 3.2. Zespoły komputera: widok z najwyższego poziomu

Moduł pamięci składa się z zestawu miejsc komórek, określonych przez sekwencyjnie ponumerowane adresy. Każde z miejsc zawiera liczbę binarną, która może być zinterpretowana albo jako rozkaz, albo jako dane. Moduł wejścia-wyjścia przenosi dane z urządzeń zewnętrznych do procesora i pamięci oraz w kierunku przeciwnym. Zawiera wewnętrzne bufory do czasowego przechowywania danych do momentu, aż będą one mogły być wysłane.

Po tym krótkim przeglądzie głównych zespołów rozpatrzymy ich współdziałanie mające na celu wykonywanie programów.

3.2. Działanie komputera

Podstawowym zadaniem komputera jest wykonywanie programu. Program przeznaczony do wykonania składa się z zestawu rozkazów przechowywanych w pamięci. Procesor realizuje tę pracę, wykonując rozkazy wyszczególnione w programie. W tym podrozdziale zostaną przedstawione podstawowe elementy wykonywania programu. W najprostszej postaci przetwarzanie rozkazu składa się z dwóch kroków: procesor odczytuje (*pobiera*) rozkaz z pamięci, a następnie wykonuje go. Realizacja programu polega na powtarzaniu procesu pobierania i wykonywania rozkazu. Wykonywanie rozkazu może zawierać pewną liczbę kroków i jest zależne od natury rozkazu (widać to np. w dolnej części rys. 2.4).



Rysunek 3.3. Podstawowy cykl rozkazu

Przetwarzanie wymagane dla pojedynczego rozkazu jest nazywane *cyklem rozkazu*. Jest on pokazany na rys. 3.3, przy czym wykorzystano 2-etapowe ujęcie przedstawione powyżej. Te dwa etapy są określane jako *cykl pobierania* i *cykl wykonywania*. Wykonywanie programu jest wstrzymywane tylko po wyłączeniu maszyny, po wystąpieniu pewnego rodzaju nieodwracalnego błędu lub jeśli wystąpi w programie rozkaz zatrzymania komputera.

Pobieranie i wykonywanie rozkazu

Na początku każdego cyklu rozkazu procesor pobiera rozkaz z pamięci. W typowym procesorze do śledzenia, który rozkaz ma być pobrany, służy rejestr zwany *licznikiem programu* (PC). Jeśli procesor nie otrzyma innego polecenia, to powoduje inkrementację (elementarny przyrost stanu) licznika PC po każdym pobraniu rozkazu i wykonuje następny rozkaz w ciągu (to znaczy rozkaz zlokalizowany w pamięci pod najbliższym adresem o kolejnym wyższym numerze). Rozpatrzmy na przykład komputer, w którym każdy rozkaz zajmuje jedno 16-bitowe słowo w pamięci. Załóżmy, że licznik programu jest ustawiony na pozycji 300. Procesor pobierze rozkaz z pozycji 300. W następnych cyklach rozkazy będą pobierane z miejsc 301, 302, 303 itd. Jak już wyjaśniliśmy, sekwencja ta może być zmieniana.

Pobrany rozkaz jest następnie ładowany do rejestru w procesorze zwanego *rejestrem rozkazu* (IR). Rozkaz ten ma postać kodu binarnego określającego działanie, które ma podjąć procesor. Procesor interpretuje rozkaz i przeprowadza wymagane działanie. Ogólnie działania te można podzielić na cztery kategorie:

- **Procesor–pamięć.** Dane mogą być przenoszone z procesora do pamięci lub z pamięci do procesora.
- **Procesor–wejście-wyjście.** Dane mogą być przenoszone z otoczenia lub do niego, przez przenoszenie ich między procesorem a modułem wejścia-wyjścia.
- **Przetwarzanie danych.** Procesor może wykonywać pewne operacje arytmetyczne lub logiczne na danych.
- **Sterowanie.** Rozkaz może określać, że sekwencja wykonywania ma być zmieniona. Na przykład procesor może pobrać rozkaz z pozycji 149, z którego wynika, że następny rozkaz ma być pobrany z pozycji 182. Procesor zapamięta ten fakt przez ustawienie licznika programu na 182. Dzięki temu w następnym cyklu pobrania rozkaz zostanie pobrany z pozycji 182, a nie 150.

Wykonywanie rozkazów może zawierać kombinacje tych działań.

Rozpatrzmy prosty przykład, posługując się hipotetyczną maszyną, której własności są przedstawione na rys. 3.4. Procesor zawiera akumulator (AC) służący do czasowego przechowywania danych. Zarówno rozkazy, jak i dane są słowami 16-bitowymi. Wygodnie jest więc zorganizować pamięć przy użyciu słów 16-bitowych. Format rozkazu zapewnia 4 bity dla kodu operacji, może więc występować $2^4 = 16$ różnych kodów operacji, a liczba słów w pamięci możliwych do bezpośredniego zaadresowania wynosi $2^{12} = 4096$ (4 K).

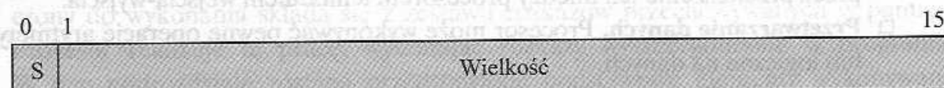
Na rysunku 3.5 jest pokazana częściowa realizacja programu z uwzględnieniem odpowiednich części pamięci i rejestrów procesora[1]. Pokazany fragment programu polega na dodaniu zawartości słowa pamięci pod adresem 940 do zawartości słowa zapisanego pod adresem 941 oraz na zapisaniu wyniku pod tym drugim adresem. Wymagane są 3 rozkazy, które mogą być opisane jako 3 cykle pobrania i 3 cykle wykonania:

1. Licznik programu zawiera liczbę 300 – adres pierwszego rozkazu. Adres ten (wartość 1940 w notacji szesnastkowej) jest ładowany do rejestru rozkazów (IR) i zawartość PC jest zwiększona o jeden. Zauważmy, że ten proces angażuje rejestr adresowy pamięci (MAR) i rejestr buforowy pamięci (MBR). Dla uproszczenia te pośrednie rejestry są pominięte.
2. Pierwsze 4 bity (pierwsza cyfra szesnastkowa) rejestru IR wskazują, że ma być ładowany akumulator (AC). Pozostałe 12 bitów (3 cyfry szesnastkowe) precyzuje adres, w tym przypadku 940, spod którego dane mają być ładowane.
3. Następna instrukcja (5941) jest pobierana z lokacji 301 i stan licznika PC jest inkrementowany.
4. Poprzednia zawartość akumulatora i zawartość pozycji 941 są dodawane, wynik zaś jest przechowywany w akumulatorze.
5. Następna instrukcja (2941) jest pobierana z lokacji 302 i stan licznika PC jest zwiększany o jeden.
6. Zawartość akumulatora jest zapisywana w pozycji 941.

[1] Użyto tutaj notacji szesnastkowej, w której każdą cyfrę reprezentują cztery bity. Jest to najwygodniejsza postać przedstawiania zawartości pamięci i rejestrów, gdy długość słowa jest wielokrotnością 4. Podstawowe informacje o systemach liczenia (dziesiętnym, binarnym, szesnastkowym) znajdują się w dodatku B.



(a) Format rozkazu



(b) Format liczby całkowitej

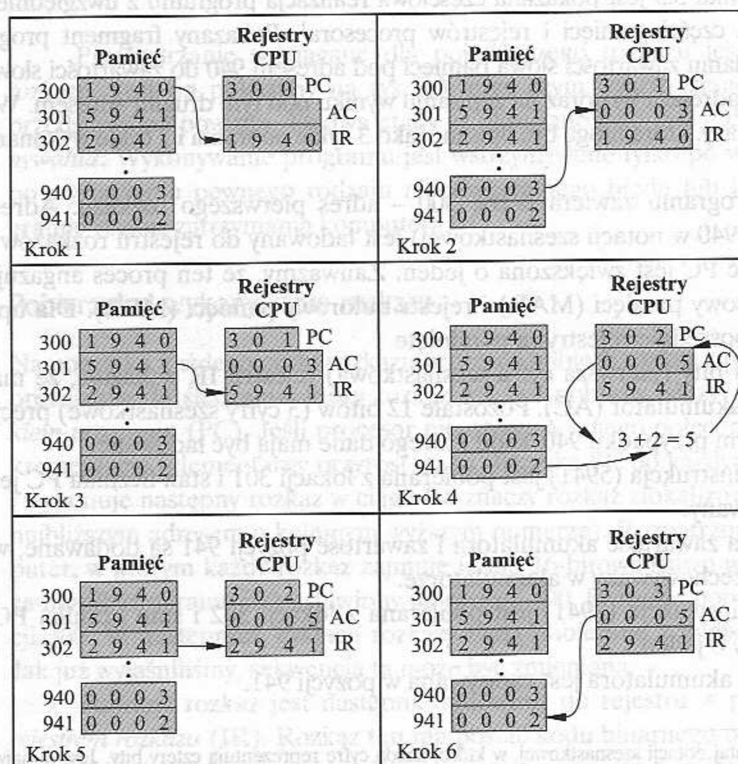
Licznik rozkazów (PC) = adres rozkazu
Rejestr rozkazów (IR) = wykonywany rozkaz
Akumulator (AC) = tymczasowe przechowywanie

(c) Wewnętrzne rejestry procesora

0001 = Ładuj AC z pamięci
0010 = Zapisz AC w pamięci
0101 = Dodaj z pamięci do AC

(d) Częściowa lista kodów operacji

Rysunek 3.4. Własności hipotetycznego komputera



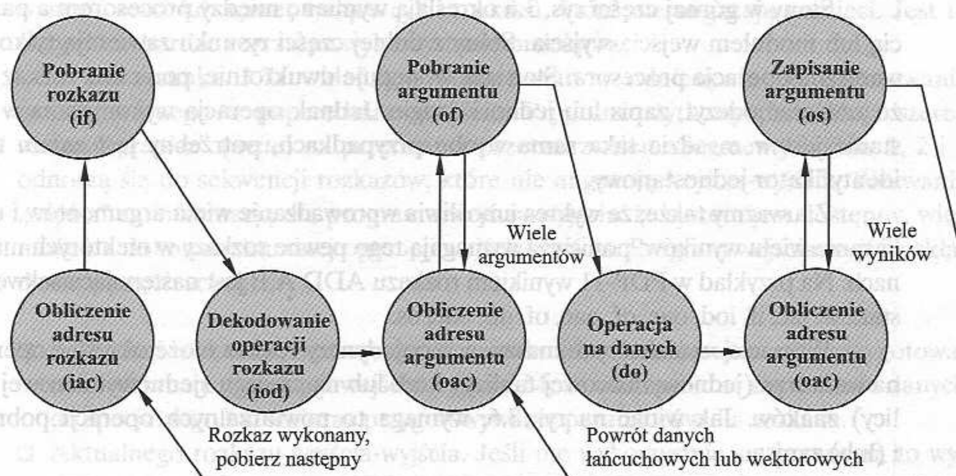
Rysunek 3.5. Przykład wykonywania programu (zawartość pamięci i rejestrów w notacji szesnastkowej)

W tym przykładzie do dodania zawartości pozycji 940 do zawartości pozycji 941 są wymagane trzy cykle rozkazu, z których każdy składa się z cyklu pobierania i cyklu wykonywania. Przy bardziej złożonym zestawie rozkazów liczba wymaganych cykli byłaby mniejsza. Na przykład niektóre starsze procesory operowały rozkazami zawierającymi więcej niż jeden adres pamięci. Cykl wykonywania określonego rozkazu w takich procesorach obejmuje więcej niż jedno odwołanie do pamięci. Zamiast odniesień do pamięci, w rozkazie może być określona operacja wejścia-wyjścia.

Na przykład rozkaz komputera PDP-11 wyrażany symbolicznie jako ADD B,A (Dodaj B,A) powoduje zapisanie sumy zawartości pamięci B i A w lokacji A. Wykonywany jest w tym celu pojedynczy cykl rozkazu zawierający następujące kroki:

- Pobranie rozkazu ADD.
- Wczytanie zawartości lokacji A z pamięci do procesora.
- Wczytanie zawartości lokacji B z pamięci do procesora. Żeby zachować zawartość A, procesor musi mieć przynajmniej dwa rejestry do przechowywania danych z pamięci, a nie tylko pojedynczy akumulator.
- Dodanie obu wartości.
- Przepisanie wyniku z procesora do lokacji pamięci A.

Tak więc cykl wykonania określonego rozkazu może wykorzystywać więcej niż jedno odniesienie do pamięci. Zamiast odnosić się do pamięci, rozkaz może precyzować operacje wejścia-wyjścia.



Rysunek 3.6. Graf stanów cyklu rozkazu

Na rysunku 3.6 jest przedstawiony dokładniejszy obraz podstawowego cyklu rozkazu z rys. 3.3, uwzględniający te dodatkowe rozważania. Rysunek ma postać wykresu stanów. Dla danego cyklu rozkazu pewne stany mogą być zerowe (mogą nie występować), inne zaś mogą występować częściej niż raz. Stany te mogą być opisane jak następuje:

- **Obliczanie adresu rozkazu (iac – *instruction address calculation*).** Określenie adresu następnego rozkazu przeznaczonego do wykonania. Zwykle polega to na dodaniu ustalonej liczby do adresu poprzedniego rozkazu. Jeśli na przykład każdy rozkaz zawiera 16 bitów, a organizacja pamięci przewiduje słowa 16-bitowe, to dodaje się 1 do poprzedniego adresu. Jeśli w pamięci przewidziano bezpośrednie adresowanie 8-bitowych bajtów, to dodaje się 2 do poprzedniego adresu.
- **Pobieranie rozkazu (if – *instruction fetch*).** Wczytanie rozkazu z pamięci do procesora.
- **Dekodowanie operacji rozkazu (iod – *instruction operation decoding*).** Analizowanie rozkazu w celu określenia rodzaju operacji, która ma być przeprowadzona, oraz argumentu (argumentów).
- **Obliczanie adresu argumentu (oac – *operand address calculation*).** Określanie adresu argumentu, jeśli operacja odnosi się do argumentu znajdującego się w pamięci lub dostępnego przez wejście-wyjście.
- **Pobieranie argumentu (of – *operand fetch*).** Pobranie argumentu z pamięci lub z wejścia-wyjścia.
- **Operacja na danych (do – *data operation*).** Przeprowadzenie operacji wskazanej w rozkazie.
- **Zapisanie argumentu (os – *operand store*).** Zapisanie wyniku w pamięci lub skierowanie go do wejścia-wyjścia.

Stany w górnej części rys. 3.6 określają wymianę między procesorem a pamięcią lub modułem wejścia-wyjścia. Stany z dolnej części rysunku zawierają tylko wewnętrzne operacje procesora. Stan oac występuje dwukrotnie, ponieważ rozkaz może zawierać odczyt, zapis lub jedno i drugie. Jednak operacja wykonywana w tym stanie jest w zasadzie taka sama w obu przypadkach, potrzebny jest zatem tylko identyfikator jednostanowy.

Zauważmy także, że wykres umożliwia wprowadzanie wielu argumentów i uzyskiwanie wielu wyników, ponieważ wymagają tego pewne rozkazy w niektórych maszynach. Na przykład w PDP-11 wynikiem rozkazu ADD A,B jest następująca sekwencja stanów: iac, if, iod, oac, of, oac, of, do, oac, os.

Wreszcie, w niektórych maszynach pojedynczy rozkaz może określać operację na wektorze (jednowymiarowej tablicy) liczb lub na szeregu (jednowymiarowej tablicy) znaków. Jak widać na rys. 3.6, wymaga to powtarzalnych operacji pobrania i (lub) zapisu.

Przerwania

Praktycznie we wszystkich komputerach przewidziano mechanizm, za pomocą którego inne moduły (wejście-wyjście, pamięć) mogą przerwać normalne przetwarzanie danych przez procesor. W tabeli 3.1 są wymienione najczęściej występujące klasy przerwań. Szczególną naturą tych przerwań zajmiemy się później, zwłaszcza w rozdz. 7 i 12. Musimy jednak wprowadzić tę koncepcję teraz, aby lepiej zrozumieć

naturę cyklu rozkazu oraz implikacje dotyczące struktury połączeń, wynikające z istnienia przerwań. Na tym etapie nie musimy być zainteresowani szczegółami generowania i przetwarzania przerwań, natomiast możemy się skupić tylko na komunikacji między modułami, wynikającej z przerwań.

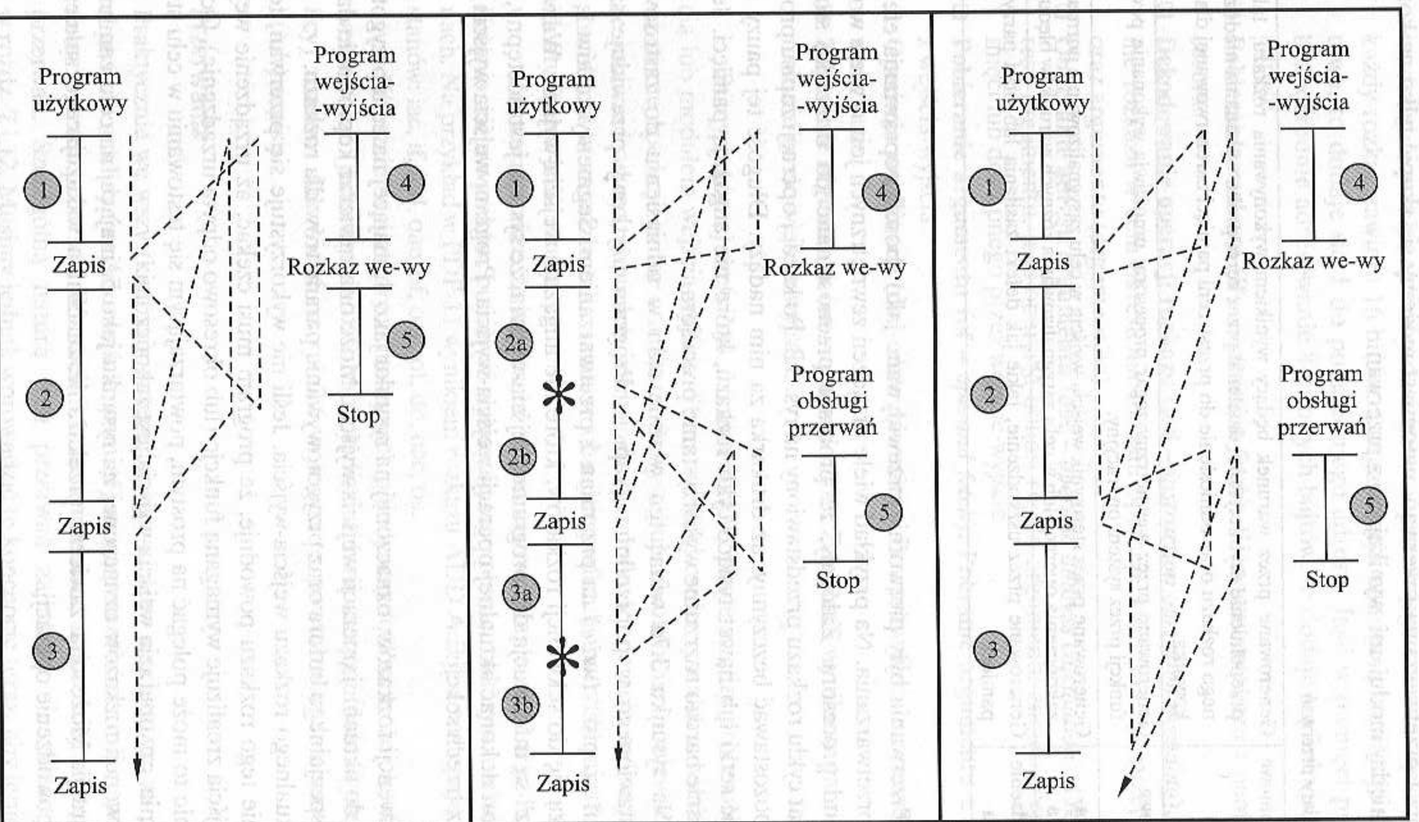
Tabela 3.1. Klasy przerwań

Programowe	Generowane przez warunek będący wynikiem wykonywania rozkazu, taki jak przepełnienie arytmetyczne, dzielenie przez zero, próba wykonania niedozwolonego rozkazu oraz odniesienie do przestrzeni pamięci zarezerwowanej dla użytkownika.
Zegarowe	Generowane przez wewnętrzny zegar procesora; umożliwia wykonanie pewnych funkcji przez system operacyjny.
Wejścia-wyjścia	Generowane przez sterownik wejścia-wyjścia w celu zasygnalizowania normalnego zakończenia operacji lub w celu zasygnalizowania różnych warunków błędu.
Uszkodzenie sprzętu	Generowane przez uszkodzenie, takie jak defekt zasilania lub błąd parzystości pamięci.

Przerwania były pierwotnie przewidywane jako sposób poprawiania efektywności przetwarzania. Na przykład wiele urządzeń zewnętrznych jest o wiele wolniejszych od procesora. Załóżmy, że procesor przenosi dane do drukarki, stosując schemat cyklu rozkazu przedstawiony na rys. 3.3. Po każdej operacji zapisu procesor musi pozostawać bezczynny, aż drukarka za nim nadąży. Długość tej pauzy może wynosić setki lub nawet tysiące cykli rozkazu, które nie angażują pamięci. Jest to oczywiście bardzo rozrzutne wykorzystanie procesora.

Na rysunku 3.7a widać ten właśnie stan w odniesieniu do zastosowania przedstawionego w poprzednim akapicie. Program użytkowy przewiduje szereg wezwań „zapisz" (*write*) na przemian z przetwarzaniem. Segmenty kodów 1, 2 i 3 odnoszą się do sekwencji rozkazów, które nie angażują wejścia-wyjścia. Wezwania „zapisz" są informacją dla programu wejścia-wyjścia, że system jest dostępny, więc może on dokonać aktualnej operacji wejścia-wyjścia. Program wejścia-wyjścia składa się z trzech sekcji:

- Sekwencji rozkazów, oznaczonej na rysunku jako 4, mającej na celu przygotowanie do aktualnej operacji wejścia-wyjścia. Może ona zawierać kopiowanie danych do specjalnego bufora oraz przygotowywanie parametrów dla rozkazu.
- Aktualnego rozkazu wejścia-wyjścia. Jeśli nie wykorzystuje się przerwań, to wydanie tego rozkazu powoduje, że program musi czekać, aż urządzenie wejścia-wyjścia zrealizuje wymaganą funkcję (lub okresowo odpyta urządzenie). Oczekiwanie to może polegać na prostym, powtarzającym się testowaniu w celu stwierdzenia, czy operacja wejścia-wyjścia jest zakończona.
- Sekwencji rozkazów oznaczonej na rysunku jako 5, mającej na celu zamknięcie operacji. Może ona zawierać przekazanie znacznika wskazującego sukces lub niepowodzenie operacji.



(a) Bez przerwań

(b) Z przerwaniami; krótkie oczekiwanie na wejście-wyjście

(c) Z przerwaniami; długie oczekiwanie na wejście-wyjście

Rysunek 3.7. Programowy przepływ sterowania bez przerwań oraz z przerwaniami

Ponieważ operacja wejścia-wyjścia może zająć stosunkowo długi czas, program wejścia-wyjścia jest zawieszany do czasu zakończenia operacji. W wyniku tego program użytkowy jest zatrzymywany na wezwaniu „zapisz" przez dość długi czas.

Przerwania i cykl rozkazu

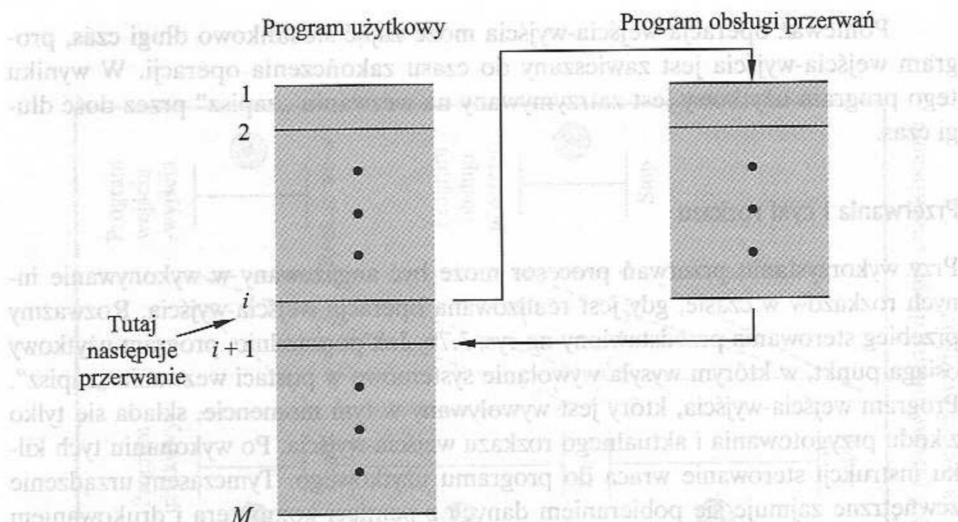
Przy wykorzystaniu przerwań procesor może być angażowany w wykonywanie innych rozkazów w czasie, gdy jest realizowana operacja wejścia-wyjścia. Rozważmy przebieg sterowania przedstawiony na rys. 3.7b. Jak poprzednio, program użytkowy osiąga punkt, w którym wysyła wywołanie systemowe w postaci wezwania „zapisz". Program wejścia-wyjścia, który jest wywoływany w tym momencie, składa się tylko z kodu przygotowania i aktualnego rozkazu wejścia-wyjścia. Po wykonaniu tych kilku instrukcji sterowanie wraca do programu użytkowego. Tymczasem urządzenie zewnętrzne zajmuje się pobieraniem danych z pamięci komputera i drukowaniem ich. Ta operacja wejścia-wyjścia jest przeprowadzana współbieżnie w stosunku do rozkazów zawartych w programie użytkowym.

Gdy urządzenie zewnętrzne staje się gotowe do obsługi, to znaczy, gdy jest gotowe do przyjmowania następnych danych z procesora, moduł wejścia-wyjścia związany z tym urządzeniem wysyła sygnał *żądania przerwania* (*interrupt request*) do procesora. Procesor odpowiada, zawieszając działanie bieżącego programu i wykonując skok do programu obsługującego to urządzenie zewnętrzne, nazywanego *programem obsługi przerwania* (*interrupt handler*). Po obsłużeniu urządzenia następuje powrót do programu bieżącego. Punkty, w których następują takie przerwania, oznaczono gwiazdkami na rys. 3.7b.

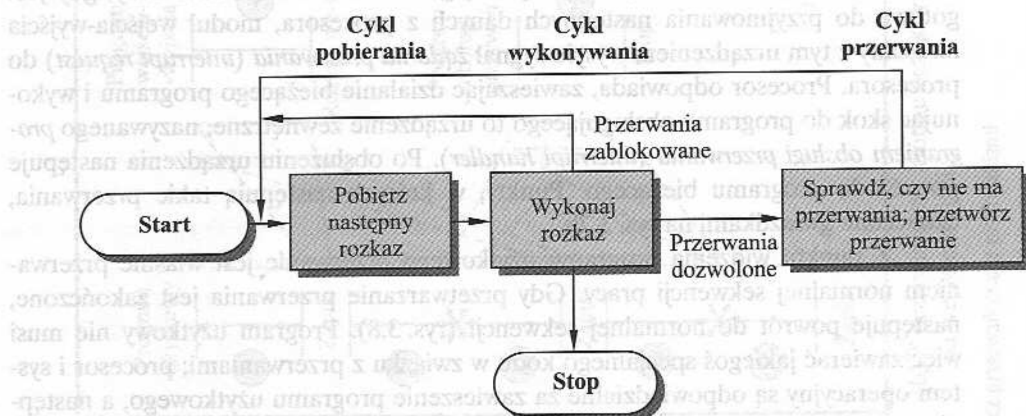
Z punktu widzenia programu użytkowego przerwanie jest właśnie przerwaniem normalnej sekwencji pracy. Gdy przetwarzanie przerwania jest zakończone, następuje powrót do normalnej sekwencji (rys. 3.8). Program użytkowy nie musi więc zawierać jakiegoś specjalnego kodu w związku z przerwaniami; procesor i system operacyjny są odpowiedzialne za zawieszenie programu użytkowego, a następnie powodują powrót do tego samego punktu.

Aby dostosować się do przerwań, do cyklu rozkazu jest dodawany *cykl przerwania* w sposób pokazany na rys. 3.9. Podczas cyklu przerwania procesor sprawdza, czy nie nastąpiły jakieś przerwania, czego świadectwem byłaby obecność sygnału przerwania. Jeśli przerwania nie są realizowane, procesor przechodzi do cyklu pobrania i pobiera następny rozkaz z bieżącego programu. Jeśli natomiast następuje przerwanie, procesor wykonuje następujące czynności:

- Zawiesza wykonywanie bieżącego programu i zachowuje jego kontekst. Polega to na zapisaniu adresu następnego rozkazu przewidywanego do wykonania (tj. bieżącej zawartości licznika programu), a także innych danych związanych z bieżącym działaniem procesora.
- Ustawia licznik programu na początkowy adres programu *obsługi przerwań*.



Rysunek 3.8. Przekazywanie sterowania za pomocą przerwań



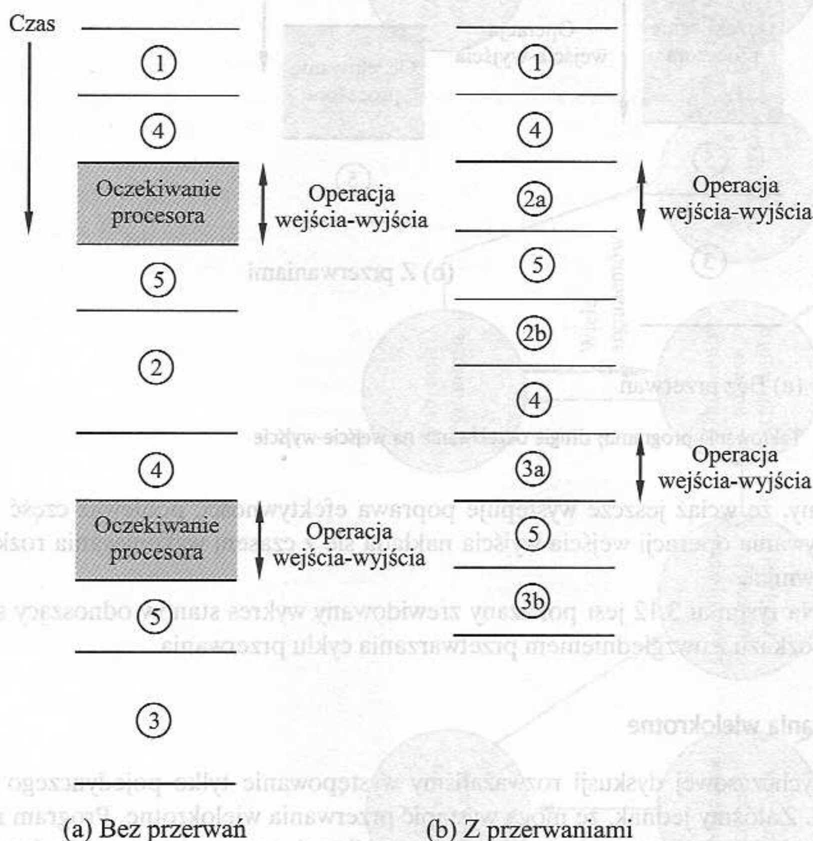
Rysunek 3.9. Cykl rozkazu z przerwaniami

Procesor przechodzi następnie do cyklu pobrania i pobiera pierwszy rozkaz z programu obsługi przerwań. Program ten jest na ogół częścią systemu operacyjnego. Zwykle program obsługi określa naturę przerwania, po czym podejmuje niezbędne działania. W użytym przez nas przykładzie program określa, który moduł wejścia-wyjścia wygenerował przerwanie, po czym może przeskoczyć do programu, który przekaże więcej danych temu modułowi. Po zakończeniu programu obsługi przerwania procesor może wznowić wykonywanie programu użytkowego w punkcie jego przerwania.

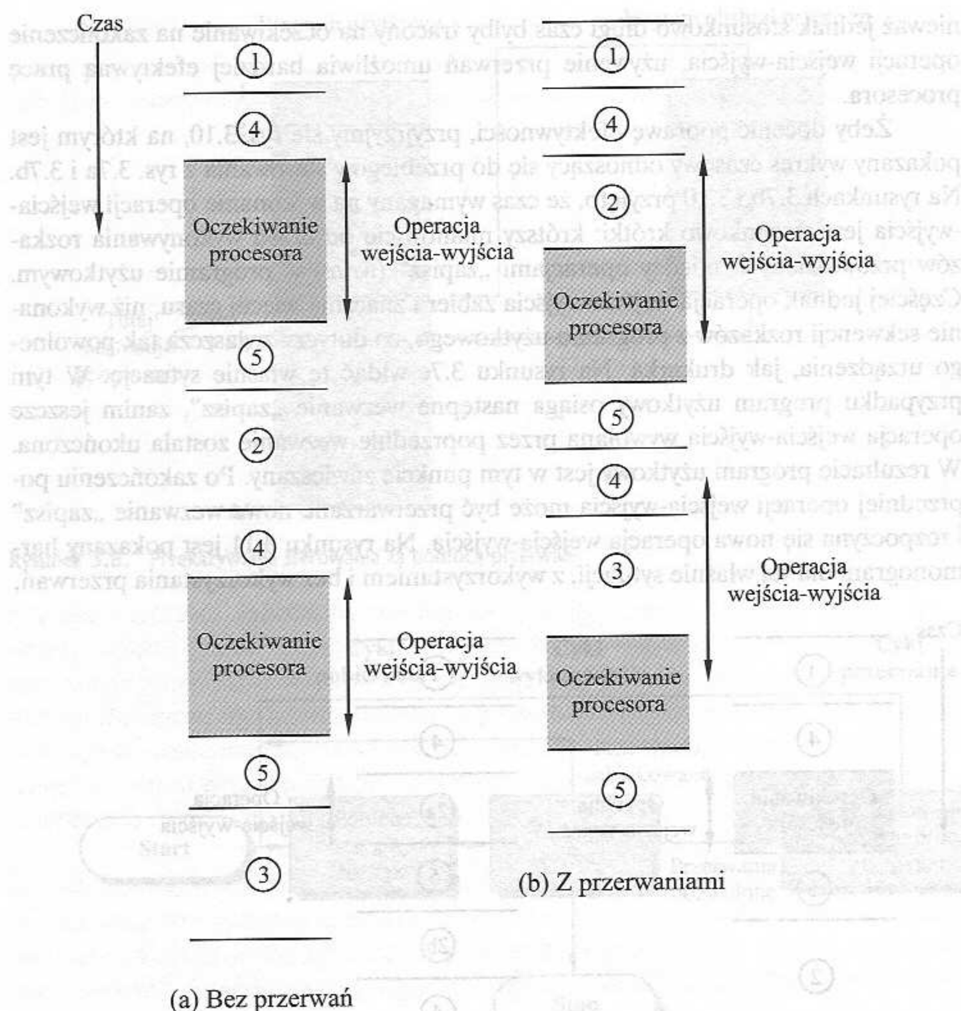
Jest jasne, że w tym procesie występują pewne dodatkowe czynności. Muszą być wykonane dodatkowe rozkazy (w ramach programu obsługi przerwań) w celu określenia natury przerwania i podjęcia decyzji w sprawie niezbędnych działań. Po-

nieważ jednak stosunkowo długi czas byłby tracony na oczekiwanie na zakończenie operacji wejścia-wyjścia, używanie przerwań umożliwia bardziej efektywną pracę procesora.

Żeby docenić poprawę efektywności, przyjrzyjmy się rys. 3.10, na którym jest pokazany wykres czasowy odnoszący się do przebiegów sterowania z rys. 3.7a i 3.7b. Na rysunkach 3.7b i 3.10 przyjęto, że czas wymagany na wykonanie operacji wejścia-wyjścia jest stosunkowo krótki: krótszy mianowicie od czasu wykonywania rozkazów przewidzianych między operacjami „zapisz" (*write*) w programie użytkowym. Częściej jednak operacja wejścia-wyjścia zabiera znacznie więcej czasu, niż wykonanie sekwencji rozkazów z programu użytkowego, co dotyczy zwłaszcza tak powolnego urządzenia, jak drukarka. Na rysunku 3.7c widać tę właśnie sytuację. W tym przypadku program użytkowy osiąga następne wezwanie „zapisz", zanim jeszcze operacja wejścia-wyjścia wywołana przez poprzednie wezwanie została ukończona. W rezultacie program użytkowy jest w tym punkcie zawieszany. Po zakończeniu poprzedniej operacji wejścia-wyjścia może być przetwarzane nowe wezwanie „zapisz" i rozpoczyna się nowa operacja wejścia-wyjścia. Na rysunku 3.11 jest pokazany harmonogram dla tej właśnie sytuacji, z wykorzystaniem i bez wykorzystania przerwań.



Rysunek 3.10. Taktowanie programu; krótkie oczekiwanie na wejście-wyjście



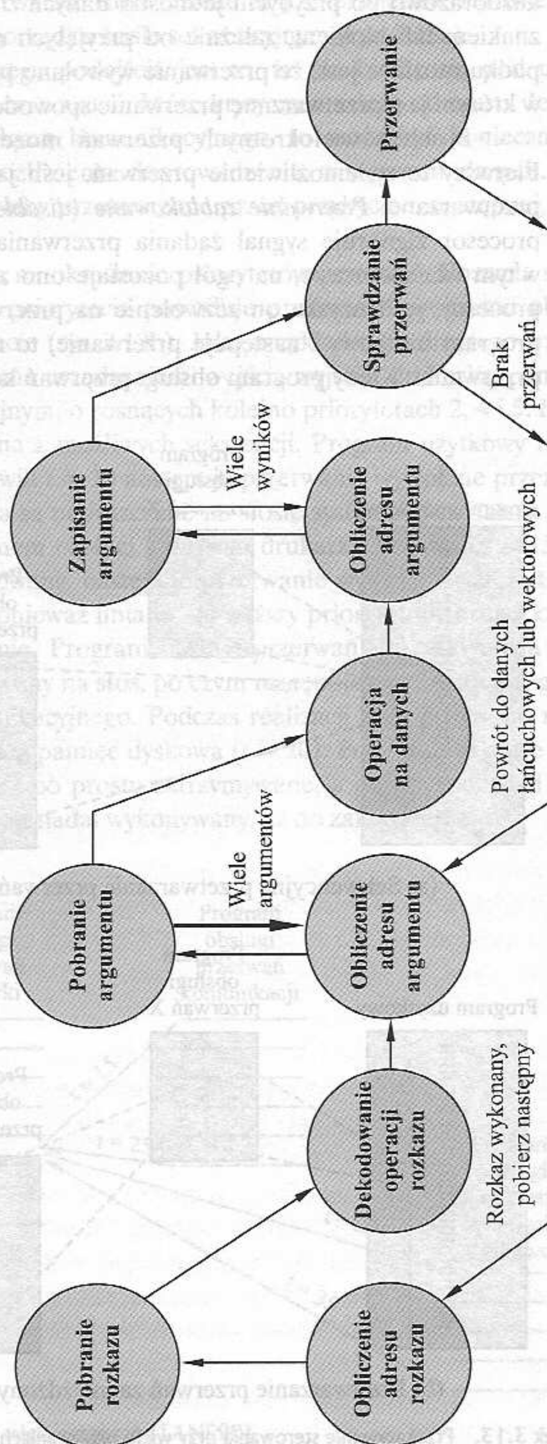
Rysunek 3.11. Taktowanie programu; długie oczekiwanie na wejście-wyjście

Widzimy, że wciąż jeszcze występuje poprawa efektywności, ponieważ część czasu wykonywania operacji wejścia-wyjścia nakłada się z czasem wykonywania rozkazów użytkownika.

Na rysunku 3.12 jest pokazany zrewidowany wykres stanów odnoszący się do cyklu rozkazu z uwzględnieniem przetwarzania cyklu przerwania.

Przerwania wielokrotne

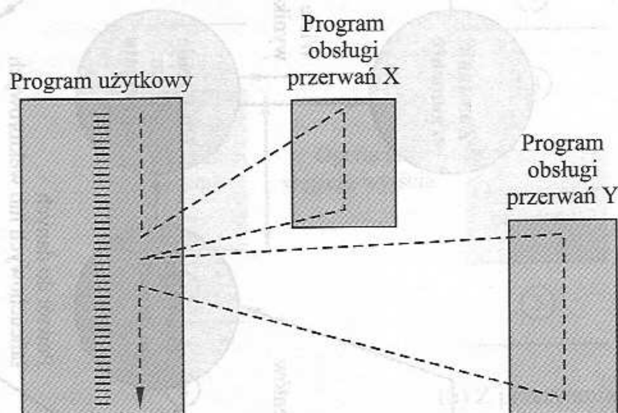
W dotychczasowej dyskusji rozważaliśmy występowanie tylko pojedynczego przerwania. Załóżmy jednak, że mogą wystąpić przerwania wielokrotne. Program może, na przykład, otrzymywać dane z łącza komunikacyjnego a także sygnały dotyczące drukowania. Drukarka będzie generowała przerwanie każdorazowo po zakończeniu



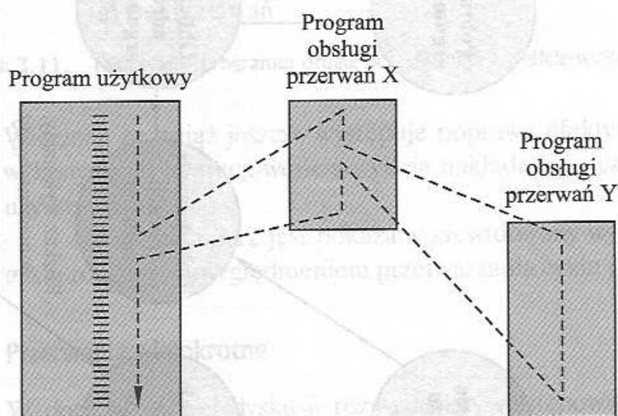
Rysunek 3.12. Graf stanów cyklu rozkazu z przerwaniami

operacji drukowania. Sterownik łącza komunikacyjnego wygeneruje przerwanie każdorazowo po przybyciu jednostki danych. Jednostka ta może być pojedynczym znakiem lub blokiem, zależnie od przyjętych reguł komunikacji. W każdym przypadku możliwe jest, że przerwanie wywołane przez ten sterownik nastąpi w czasie, w którym jest przetwarzane przerwanie spowodowane przez drukarkę.

Problem wielokrotnych przerwań może być rozwiązany na dwa sposoby. Pierwszy to uniemożliwienie przerwań, jeśli jakiekolwiek przerwanie jest właśnie przetwarzane. *Przerwanie zablokowane* (*disabled interrupt*) oznacza po prostu, że procesor zignoruje sygnał żądania przerwania. Gdy żądanie przerwania nastąpi w tym właśnie czasie, na ogół pozostaje ono zawieszone i zostanie wykryte przez procesor, jeśli uzyska on zezwolenie na przerwanie. Jeżeli jest więc wykonywany program użytkowy i następuje przerwanie, to natychmiast uniemożliwiane są inne przerwania. Kiedy program obsługi przerwań kończy działanie, przerwania stają się



(a) Sekwencyjne przetwarzanie przerwań



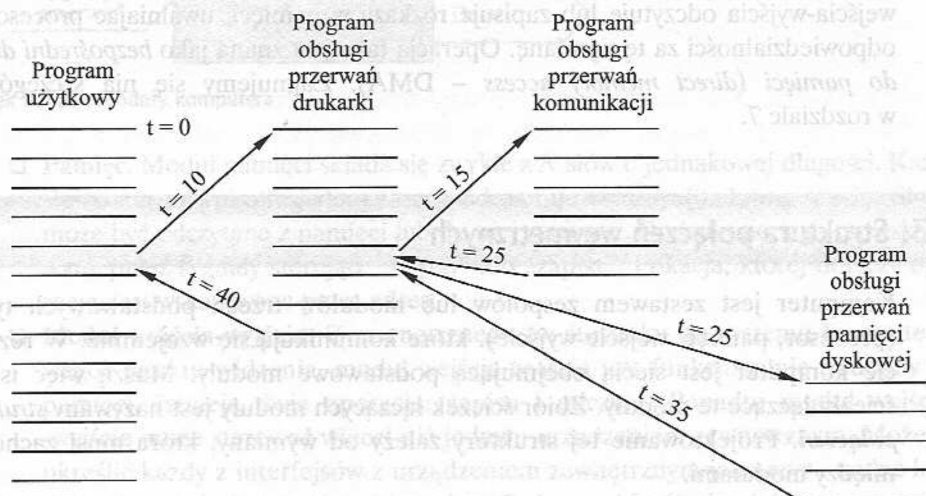
(b) Przetwarzanie przerwań zagnieżdżonych

Rysunek 3.13. Przekazywanie sterowania przy wielu przerwaniach

dozwolone jeszcze przed wznowieniem programu użytkowego, a procesor sprawdza, czy wystąpiły dodatkowe przerwania. Rozwiązanie to jest przyjemne i proste, jeśli przerwania są realizowane w porządku ściśle sekwencyjnym (rys. 3.13a).

Ujemną stroną powyższego podejścia jest to, że nie bierze się pod uwagę względnej ważności przerwań oraz zadań, które krytycznie zależą od czasu. Jeśli na przykład przybywają dane z łącza komunikacyjnego, to może być konieczne ich szybkie przyjęcie w celu umożliwienia doprowadzenia następnych danych. Jeśli pierwsza porcja danych nie byłaby przetworzona przed przybyciem następnej, dane mogłyby zostać utracone.

Drugie podejście polega na określeniu priorytetów przerwań. Pozwala się na to, że przerwanie o wyższym priorytecie powoduje przerwanie programu obsługi przerwania o niższym priorytecie (rys. 3.13b). Jako przykład tego drugiego rozwiązania, rozważmy system z trzema urządzeniami wejścia-wyjścia: drukarką, pamięcią dyskową i łączem komunikacyjnym, o rosnących kolejno priorytetach 2, 4 i 5. Na rysunku 3.14 jest pokazana jedna z możliwych sekwencji. Program użytkowy rozpoczyna się w chwili $t = 0$. W chwili $t = 10$ następuje przerwanie wywołane przez drukarkę; informacje użytkownika są umieszczane na stosie systemowym, dalsza praca odbywa się zgodnie z programem obsługi przerwań drukarki. W chwili $t = 15$, gdy program ten jest nadal realizowany, następuje przerwanie wywołane przez sterownik łącza komunikacyjnego. Ponieważ linia ta ma wyższy priorytet niż drukarka, honorowane jest nowe przerwanie. Program obsługi przerwań drukarki ulega przerwaniu, stan danych jest kierowany na stos, po czym następuje wykonanie programu obsługi przerwań łącza komunikacyjnego. Podczas realizacji tego programu następuje przerwanie wywołane przez pamięć dyskową ($t = 20$). Ponieważ to nowe przerwanie ma niższy priorytet, jest po prostu zatrzymywane, a program obsługi przerwań łącza komunikacyjnego jest nadal wykonywany, aż do zakończenia.



Rysunek 3.14. Przykład sekwencji czasowej wielu przerwań [TANE90]

Po zakończeniu programu obsługi przerwań łącza komunikacyjnego ($t = 25$), odtwarzany jest poprzedni stan procesora, w którym wykonywany jest program obsługi przerwań drukarki. Zanim jednak może być wykonany pierwszy rozkaz z tego programu, procesor honoruje mające wyższy priorytet przerwanie pamięci dyskowej, a sterowanie ulega przeniesieniu do programu obsługi przerwań pamięci dyskowej. Dopiero po zakończeniu tego programu ($t = 35$) wznawiany jest program obsługi przerwań drukarki. Kiedy z kolei ten program jest zakończony ($t = 40$), sterowanie powraca w końcu do programu użytkowego.

Działanie modułów wejścia-wyjścia

Dotychczas rozważaliśmy działanie komputera jako sterowane przez jednostkę centralną, skupiając się przede wszystkim na współpracy procesora i pamięci. Czyniliśmy zaledwie aluzje do roli modułów wejścia-wyjścia. Zajmiemy się tym szczegółowo w rozdz. 7, tutaj przedstawimy je jedynie pokrótce.

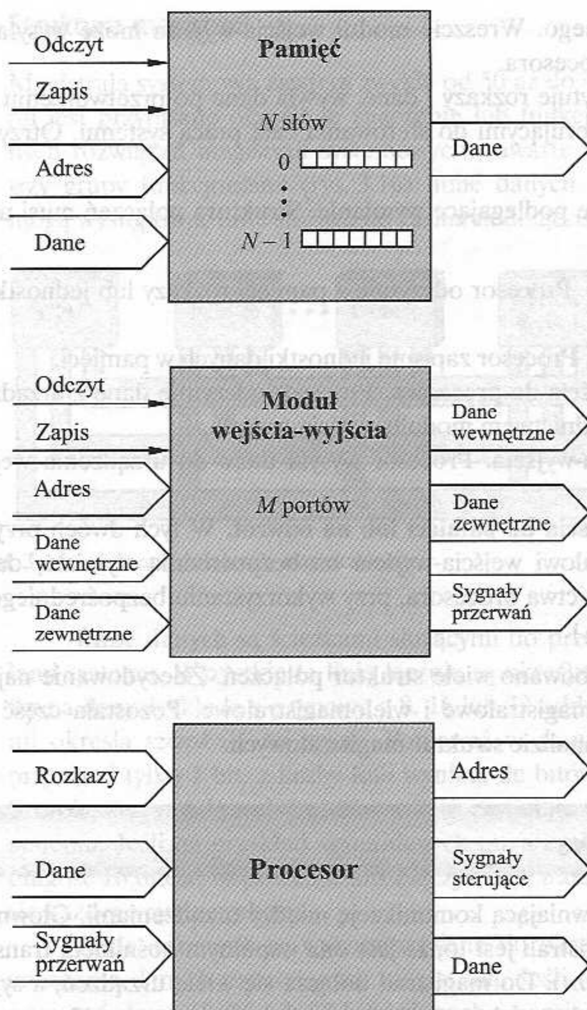
Moduł wejścia-wyjścia (np. sterownik dysku) może wymieniać dane bezpośrednio z procesorem. Podobnie jak procesor może inicjować odczyt lub zapis w pamięci, określając adres specyficznej lokacji, może on też odczytywać dane pochodzące z modułu wejścia-wyjścia lub je w nim zapisywać. W tym ostatnim przypadku procesor identyfikuje urządzenie, którym steruje określony moduł wejścia-wyjścia. Następnie może być realizowana sekwencja rozkazów o formie podobnej do tej z rys. 3.5, przy czym rozkazy odnoszą się do wejścia-wyjścia zamiast do pamięci.

W pewnych przypadkach pożądane jest umożliwienie bezpośredniej wymiany danych między wejściem-wyjściem a pamięcią. W takim przypadku procesor przekazuje modułowi wejścia-wyjścia prawo do odczytywania lub zapisywania rozkazów w pamięci, dzięki czemu przesyłanie danych między wejściem-wyjściem a pamięcią może następować bez angażowania procesora. Podczas takiego przesyłania moduł wejścia-wyjścia odczytuje lub zapisuje rozkazy w pamięci, uwalniając procesor od odpowiedzialności za tę wymianę. Operacja taka jest znana jako *bezpośredni dostęp do pamięci* (*direct memory access* – DMA). Zajmujemy się nią szczegółowo w rozdziale 7.

3.3. Struktura połączeń wewnętrznych

Komputer jest zestawem zespołów lub modułów trzech podstawowych typów (procesor, pamięć, wejście-wyjście), które komunikują się wzajemnie. W rezultacie komputer jest siecią obejmującą podstawowe moduły. Muszą więc istnieć ścieżki łączące te moduły. Zbiór ścieżek łączących moduły jest nazywany *strukturą połączeń*. Projektowanie tej struktury zależy od wymiany, która musi zachodzić między modułami.

Na rysunku 3.15 są podane rodzaje wymiany, jakie mogą być potrzebne, z uwzględnieniem podstawowych form wejścia i wyjścia dla każdego typu modułu.



Rysunek 3.15. Moduły komputera

- **Pamięć.** Moduł pamięci składa się zwykle z N słów o jednakowej długości. Każde słowo ma przypisany jednoznaczny adres numeryczny (0, 1, ..., $N-1$). Słowo może być odczytane z pamięci lub do niej zapisane. Rodzaj operacji jest wskazywany przez sygnały sterujące „czytaj" lub „zapisz". Lokacja, której dotyczy operacja, jest wskazywana przez adres.
- **Moduł wejścia-wyjścia.** Z zewnętrznego (w stosunku do systemu komputerowego) punktu widzenia, moduł wejścia-wyjścia jest funkcjonalnie podobny do pamięci. Istnieją dwie operacje: zapisu i odczytu. Ponadto moduł wejścia-wyjścia może sterować więcej niż jednym urządzeniem zewnętrznym. Możemy określić każdy z interfejsów z urządzeniem zewnętrznym jako *port* i nadać każdemu z nich jednoznaczny adres (np. 0, 1, ..., $M-1$). Istnieją poza tym zewnętrzne ścieżki danych służące do wprowadzania i wyprowadzania danych

z urządzenia zewnętrznego. Wreszcie moduł wejścia-wyjścia może wysyłać sygnały przerwania do procesora.

- **Procesor.** Procesor wczytuje rozkazy i dane, wysyła dane po przetworzeniu i posługuje się sygnałami sterującymi do sterowania całą pracą systemu. Otrzymuje też sygnały przerwania.

Powyższa lista określa dane podlegające wymianie. Struktura połączeń musi umożliwiać przesyłanie danych:

- **Z pamięci do procesora.** Procesor odczytuje z pamięci rozkazy lub jednostki danych.
- **Z procesora do pamięci.** Procesor zapisuje jednostki danych w pamięci.
- **Z urządzeń wejścia-wyjścia do procesora.** Procesor odczytuje dane z urządzenia wejścia-wyjścia za pośrednictwem modułu wejścia-wyjścia.
- **Z procesora do wejścia-wyjścia.** Procesor wysyła dane do urządzenia wejścia-wyjścia.
- **Z urządzeń wejścia-wyjścia do pamięci lub na odwrót.** W tych dwóch przypadkach zezwala się modułowi wejścia-wyjścia na bezpośrednią wymianę danych z pamięcią, bez pośrednictwa procesora, przy wykorzystaniu bezpośredniego dostępu do pamięci (DMA).

W przeszłości wypróbowano wiele struktur połączeń. Zdecydowanie najpowszechniejsze są struktury magistralowe i wielomagistralowe. Pozostała część tego rozdziału jest poświęcona analizie struktur magistralowych.

3.4. Połączenia magistralowe

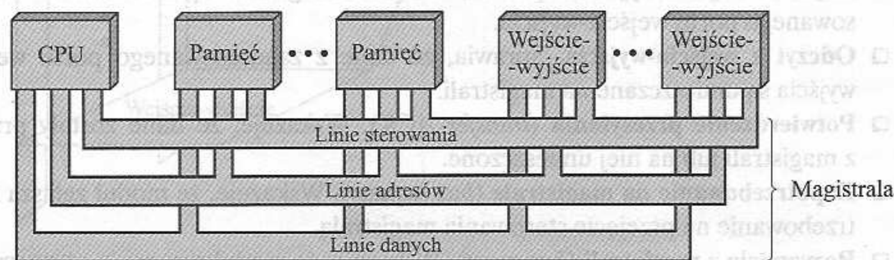
Magistrala jest drogą zapewniającą komunikację między urządzeniami. Główną cechą charakterystyczną magistrali jest to, że jest ona wspólnym nośnikiem transmisji (*shared transmission medium*). Do magistrali dołącza się wiele urządzeń, a sygnały wysyłane przez którekolwiek z nich mogą być odbierane przez wszystkie pozostałe urządzenia. Jeśli dwa urządzenia nadawałyby w tym samym czasie, ich sygnały nakładałyby się i ulegały zakłócaniu. W określonym czasie może więc nadawać tylko jedno urządzenie.

Często magistrala składa się z wielu dróg (linii) komunikacyjnych. Każdą linią mogą być przesyłane sygnały reprezentujące binarne 0 i 1. W ciągu pewnego czasu przez pojedynczą linię może być przekazana sekwencja cyfr binarnych. Kilka linii zawartych w magistrali można wykorzystywać razem do jednoczesnego (równoległego) transmitowania cyfr binarnych. Na przykład 8-bitowa jednostka danych może być przesyłana przez 8 linii magistrali.

System komputerowy zawiera pewną liczbę różnych magistrali, które łączą zespoły komputera na różnych poziomach hierarchii. Magistrala łącząca główne zespoły komputera (procesor, pamięć, wejście-wyjście) jest nazywana *magistralą systemową*. Najczęściej spotykane struktury połączeń komputera wykorzystują jedną lub więcej magistrali systemowych.

Struktura magistrali

Magistrala systemowa zawiera zwykle od 50 aż do setek oddzielnych linii. Każdej linii jest przypisane określone znaczenie lub funkcja. Chociaż występuje wiele różnych rozwiązań magistrali systemowych, zawarte w nich linie można podzielić na trzy grupy funkcjonalne (rys. 3.16): linie danych, adresów i sterowania. Ponadto mogą występować linie służące do zasilania dołączonych modułów.



Rysunek 3.16. Schemat połączenia magistralowego

Linie danych są ścieżkami służącymi do przenoszenia danych między modułami systemu. Wszystkie te linie łącznie są określane jako *szyna danych* (*data bus*). Szyna danych składa się typowo z 8, 16 lub 32 oddzielnych linii, przy czym liczba linii określa *szerokość* tej szyny. Ponieważ w danym momencie każda linia może przenosić tylko 1 bit, z liczby linii wynika, ile bitów można jednocześnie przenosić. Szerokość szyny danych jest kluczowym czynnikiem określającym wydajność całego systemu. Jeśli na przykład szyna danych ma szerokość 8 bitów, a każdy rozkaz ma długość 16 bitów, to procesor musi łączyć się z modułem pamięci dwukrotnie w czasie każdego cyklu rozkazu.

Linie adresowe są wykorzystywane do określania źródła lub miejsca przeznaczenia danych przesyłanych magistralą. Jeśli na przykład procesor ma zamiar odczytać słowo (8, 16 lub 32 bity) danych z pamięci, umieszcza adres potrzebnego słowa na linii adresowej. Jest jasne, że szerokość szyny adresowej determinuje maksymalną możliwą pojemność pamięci systemu. Ponadto linie adresowe są również używane do adresowania portów wejścia-wyjścia. Najczęściej najbardziej znaczące bity służą do wybrania określonego modułu na magistrali, natomiast najmniej znaczące bity określają lokację w pamięci lub port wejścia-wyjścia wewnątrz modułu. W przypadku szyny 8-bitowej adres 01111111 i niższe mogą na przykład oznaczać lokacje w module pamięci (moduł 0) z 128 słowami pamięci, a adres 10000000 i wyższe odnoszą się do urządzeń dołączonych do modułu wejścia-wyjścia (moduł 1).

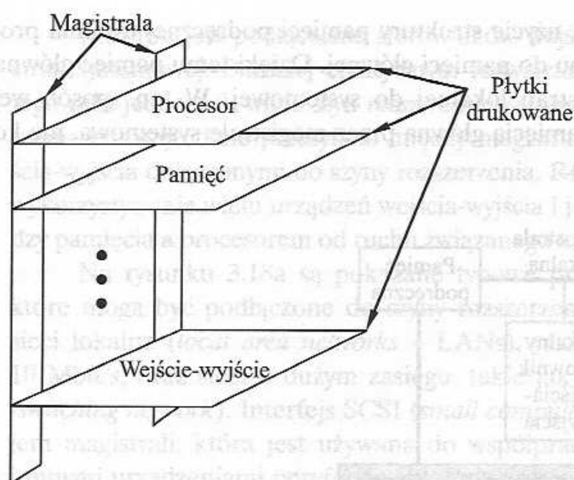
Linii sterowania używa się do sterowania dostępem do linii danych i linii adresowych, a także do sterowania ich wykorzystaniem. Ponieważ linie danych i adresowe służą wszystkim zespołom, musi istnieć sposób sterowania ich używaniem. Sygnały sterujące przekazywane między modułami systemu zawierają zarówno rozkazy, jak i informacje regulujące czas (taktujące). Sygnały czasowe określają ważność da-

nych i adresów. Sygnały rozkazów precyzują operacje, które mają być przeprowadzone. Typowe linie sterowania to:

- **Zapis w pamięci.** Sprawia, że dane z magistrali zostają zapisane pod określonym adresem.
- **Odczyt z pamięci.** Sprawia, że dane spod określonego adresu są umieszczane w magistrali.
- **Zapis do wejścia-wyjścia.** Sprawia, że dane z magistrali są kierowane do zaadresowanego portu wejścia-wyjścia.
- **Odczyt z wejścia-wyjścia.** Sprawia, że dane z zaadresowanego portu wejścia-wyjścia są umieszczane na magistrali.
- **Potwierdzenie przesyłania** (*transfer ACK*). Wskazuje, że dane zostały przyjęte z magistrali lub na niej umieszczone.
- **Zapotrzebowanie na magistralę** (*bus request*). Wskazuje, że moduł zgłasza zapotrzebowanie na przejęcie sterowania magistralą.
- **Rezygnacja z magistrali** (*bus grant*). Wskazuje, że moduł rezygnuje ze sterowania magistralą.
- **Żądanie przerwania** (*interrupt request*). Wskazuje, że przerwanie jest zawieszone.
- **Potwierdzenie przerwania** (*interrupt ACK*). Potwierdza, że zawieszone przerwanie zostało rozpoznane.
- **Zegar** (*clock*). Wykorzystywany do synchronizowania operacji.
- **Przywrócenie** (*reset*). Ustawia wszystkie moduły w stanie początkowym.

Działanie magistrali jest następujące. Jeśli jeden z modułów zamierza wysłać dane do drugiego, to musi wykonać dwie rzeczy: (1) uzyskać dostęp do magistrali i (2) przekazać dane za pośrednictwem magistrali. Jeśli natomiast zamierza uzyskać dane z innego modułu, to musi: (1) uzyskać dostęp do magistrali i (2) przekazać zapotrzebowanie do tego modułu przez odpowiednie linie sterowania i adresowe. Musi następnie czekać, aż drugi moduł wyśle dane.

Fizycznie magistrala systemowa jest zbiorem równoległych połączeń elektrycznych. Połączenia te są ścieżkami wytrawionymi w obwodzie drukowanym. Magistrala rozciąga się przez cały system, a wszystkie jego zespoły są podłączone do pewnej liczby lub do wszystkich linii magistrali. Bardzo powszechne rozwiązanie fizyczne jest przedstawione na rys. 3.17. W przykładzie tym magistrala składa się z dwóch pionowych zespołów połączeń. Wzdłuż tych zespołów połączeń, w regularnych odstępach są rozmieszczone punkty mocowania w postaci poziomych gniazd podtrzymujących płytki drukowane. Każdy z głównych zespołów systemu zajmuje jedną lub więcej takich płytek i jest połączony z magistralą poprzez gniazda. Cały ten zestaw jest zamknięty w obudowie. Układ taki nadal może być stosowany w niektórych magistralach systemów komputerowych. Jednak w nowoczesnych systemach występuje tendencja do umieszczania wszystkich głównych składników na tej samej płycie, przy czym coraz więcej elementów znajduje się w tym samym mikroukładzie co procesor. Zatem szyna wbudowana w mikroukład może łączyć procesor i pamięć podręczną, podczas gdy szyna znajdująca się na płycie może łączyć procesor z pamięcią główną i innymi zespołami.



Rysunek 3.17. Typowa realizacja fizyczna architektury magistralowej

Przedstawione rozwiązanie jest najwygodniejsze. Można zakupić mały system komputerowy a następnie go rozszerzyć (zwiększyć pamięć, rozbudować wejście-wyjście) przez dołączenie dodatkowych modułów. Jeśli element modułu ulegnie uszkodzeniu, można łatwo usunąć i wymienić ten moduł.

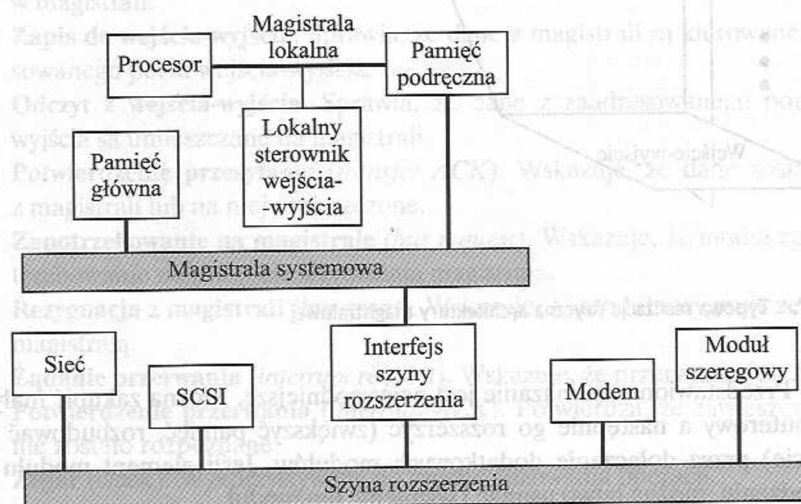
Hierarchiczne struktury wielomagistralowe

Jeśli do magistrali jest dołączona znaczna liczba przyrządów, to cierpi na tym wydajność. Są tego dwie główne przyczyny:

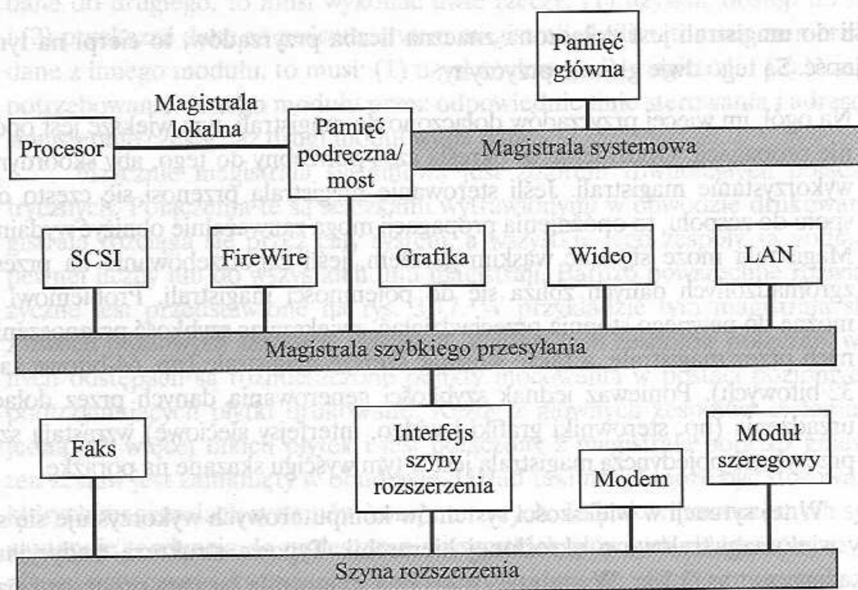
1. Na ogół, im więcej przyrządów dołączono do magistrali, tym większe jest opóźnienie propagacji. Opóźnienie to określa czas potrzebny do tego, aby skoordynować wykorzystanie magistrali. Jeśli sterowanie magistralą przenosi się często od zespołu do zespołu, to opóźnienia propagacji mogą zauważalnie obniżyć wydajność.
2. Magistrala może się stać wąskim gardłem, jeśli zapotrzebowanie na przesłanie zgromadzonych danych zbliża się do pojemności magistrali. Problemowi temu można do pewnego stopnia przeciwdziałać, zwiększając szybkość przenoszenia danych przez magistralę, a także stosując szersze magistrale (np. 64-bitowe zamiast 32-bitowych). Ponieważ jednak szybkości generowania danych przez dołączone urządzenia (np. sterowniki grafiki i wideo, interfejsy sieciowe) wzrastają szybko, przesyłanie pojedynczą magistralą jest w tym wyścigu skazane na porażkę.

W tej sytuacji w większości systemów komputerowych wykorzystuje się struktury wielomagistralowe o określonej hierarchii. Typowa struktura tradycyjna jest pokazana na rys. 3.18a. Występuje tu lokalna magistrala łącząca procesor i pamięć podręczną. Może ona wspomagać jedno lub więcej urządzeń lokalnych. Sterownik pamięci podręcznej łączy tę pamięć nie tylko do magistrali lokalnej, ale również do magistrali systemowej, do której są dołączone wszystkie moduły pamięci głównej.

Jak dowiemy się w rozdz. 4, użycie struktury pamięci podręcznej uwalnia procesor od potrzeby częstego dostępu do pamięci głównej. Dzięki temu pamięć główna może być przesunięta z magistrali lokalnej do systemowej. W ten sposób wejście-wyjście komunikuje się z pamięcią główną przez magistralę systemową, nie kolidując z działalnością procesora.



(a) Tradycyjna architektura magistralowa



(b) Architektura o dużej wydajności

Rysunek 3.18. Przykładowe konfiguracje magistralowe

Możliwe jest podłączenie sterowników wejścia-wyjścia bezpośrednio do magistrali systemowej. Bardziej efektywnym rozwiązaniem jest jednak wykorzystanie do tego celu jednej lub wielu *szyn rozszerzenia* (*expansion buses*). Interfejs szyny rozszerzenia buforuje dane przesyłane między magistralą systemową a sterownikami wejścia-wyjścia dołączonymi do szyny rozszerzenia. Rozwiązanie to umożliwia systemowi wykorzystywanie wielu urządzeń wejścia-wyjścia i jednocześnie izolowanie ruchu między pamięcią a procesorem od ruchu związanego z wejściem-wyjściem.

Na rysunku 3.18a są pokazane typowe przykłady urządzeń wejścia-wyjścia, które mogą być podłączone do szyny rozszerzenia. Połączenia sieciowe obejmują sieci lokalne (*local area networks* – LANs), takie jak Ethernet o przepustowości 10 Mbit/s, oraz sieci o dużym zasięgu, takie jak sieć komutacji pakietów (*packet-switching network*). Interfejs SCSI (*small computer system interface*) sam jest rodzajem magistrali, która jest używana do współpracy z lokalnymi napędami dysków i innymi urządzeniami peryferyjnymi. Port szeregowy może być wykorzystywany do współpracy z drukarką lub skanerem.

Ta tradycyjna architektura magistralowa jest rozsądnie efektywna, jednak traci na powodzeniu w miarę, jak rośnie wydajność urządzeń wejścia-wyjścia. W odpowiedzi na to zapotrzebowanie powszechnym rozwiązaniem przyjmowanym w przemyśle jest budowanie szybkich magistrali ściśle zintegrowanych z resztą systemu, wymagających tylko mostu między magistralą procesora a magistralą szybką. Rozwiązanie to jest czasem określane jako *architektura międzypiętrowa* (*mezzanine architecture*).

Na rysunku 3.18b jest pokazana typowa realizacja powyższego rozwiązania. Znów występuje tu magistrala lokalna, łącząca procesor ze sterownikiem pamięci podręcznej, który z kolei jest podłączony do magistrali systemowej współpracującej z pamięcią główną. Sterownik pamięci podręcznej jest zintegrowany z mostem (urządzeniem buforującym) łączącym z magistralą szybką. Magistrala ta obsługuje połączenia z szybkimi sieciami LAN, takimi jak Fast Ethernet o przepustowości 100 Mbit/s, sterowniki urządzeń graficznych i wideo oraz sterowniki interfejsów z lokalnymi magistralami peryferyjnymi, w tym SCSI i FireWire. Ten ostatni jest urządzeniem z szybką magistralą, zaprojektowanym specjalnie do obsługi urządzeń wejścia-wyjścia o dużej przepustowości. Wolniejsze urządzenia nadal są obsługiwane przez szynę rozszerzenia z interfejsem buforującym ruch między szyną rozszerzenia a szybką magistralą.

Zaletą tego rozwiązania jest to, że szybka magistrala ściślej sprzęga procesor z urządzeniami wejścia-wyjścia o wysokich wymaganiach, a jednocześnie jest niezależna od procesora. Dzięki temu mogą być tolerowane różnice szybkości procesora i szybkiej magistrali, a także różne definicje linii sygnałowych. Zmiany architektury procesora nie wpływają na szybką magistralę i vice versa.

Elementy projektowania magistrali

Chociaż występuje wiele różnych rozwiązań magistrali, istnieje kilka podstawowych parametrów i elementów projektowych, które służą do klasyfikowania i różnicowania magistrali. W tabeli 3.2 są wymienione główne z tych elementów.

Tabela 3.2. Elementy projektowania magistrali

Rodzaj specjalistyczna multipleksowana	**Szerokość magistrali** adres dane
Metoda arbitrażu scentralizowana rozproszona	**Rodzaj transferu danych** odczyt zapis odczyt-modyfikacja-zapis odczyt po zapisie blokowy
Koordynacja czasowa synchroniczna asynchroniczna	

Rodzaje magistrali

Linie magistralowe mogą być podzielone na dwa rodzaje: specjalistyczne (*dedicated*) i multipleksowane. Linia specjalistyczna jest trwale przypisana albo jednej funkcji, albo fizycznie określonym zespołom komputera.

Przykładem specjalizacji funkcjonalnej jest zastosowanie oddzielnych, specjalistycznych linii adresów i danych, co jest powszechne w wielu magistralach. Nie jest to jednak jedyne rozwiązanie. Na przykład adresy i dane mogą być transmitowane przez ten sam zespół linii przy wykorzystaniu linii sterowania określającej *ważność adresu* (*adress valid*). W tym przypadku każdy moduł ma do dyspozycji określony odcinek czasu na skopiowanie adresu i stwierdzenie, czy jest modułem adresowanym. Adres jest następnie usuwany z magistrali, a te same połączenia magistralowe są wykorzystywane do przenoszenia danych odczytywanych lub zapisywanych. Ta właśnie metoda używania tych samych linii do wielu celów jest znana jako *multipleksowanie czasowe* (*time multiplexing*).

Zaletą multipleksowania czasowego jest stosowanie mniejszej liczby linii, co pozwala oszczędzić miejsce i (zwykle) koszt. Wadą jest to, że wewnątrz każdego modułu są potrzebne bardziej złożone układy. Potencjalnie może też nastąpić zmniejszenie wydajności, ponieważ niektóre zdarzenia wymagające tych samych linii nie mogą zachodzić równolegle.

Specjalizacja fizyczna (*physical dedication*) odnosi się do używania wielu magistrali, z których każda łączy tylko określoną grupę modułów. Typowym przykładem jest zastosowanie magistrali wejścia-wyjścia do łączenia wszystkich modułów wejścia-wyjścia. Magistrala ta jest następnie łączona z magistralą główną za pomocą pewnego rodzaju adaptacyjnego modułu wejścia-wyjścia. Potencjalną zaletą specjalizacji fizycznej jest duża przepustowość, ponieważ wypełnienie magistrali jest mniejsze. Wadą jest zwiększony rozmiar i koszt systemu.

Metoda arbitrażu

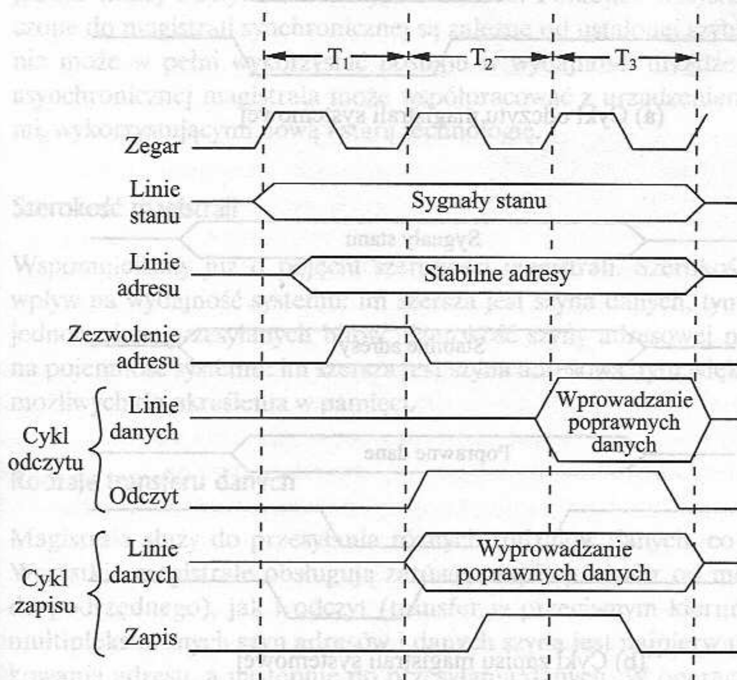
We wszystkich systemach z wyjątkiem najprostszych więcej niż jeden moduł może potrzebować przejęcia sterowania magistralą. Na przykład moduł wejścia-wyjścia może wymagać odczytu lub zapisu bezpośrednio w pamięci, bez wysyłania danych

do procesora. Ponieważ w określonym czasie tylko jeden moduł może transmitować przez magistralę, potrzebna jest jakaś metoda arbitrażu. Możliwe metody można z grubsza podzielić na scentralizowane i rozproszone. W układzie scentralizowanym istnieje jedno urządzenie, zwane *sterownikiem magistrali* lub *arbitrem*, które jest odpowiedzialne za gospodarowanie czasem na magistrali. Urządzenie to może być oddzielnym modułem lub częścią procesora. W układzie rozproszonym centralny sterownik nie występuje. Każdy moduł zawiera układy logiczne sterujące dostępem, a moduły współpracują, korzystając ze wspólnej magistrali. W obu metodach arbitrażu celem jest wyznaczenie jednego urządzenia – albo procesora, albo modułu wejścia-wyjścia – jako nadrzędnego. Urządzenie *nadrzędne* (*master*) może następnie inicjować transfer danych z innym urządzeniem, które w tym określonym przesyłaniu gra rolę *podrzędną* (*slave*).

Koordynacja czasowa

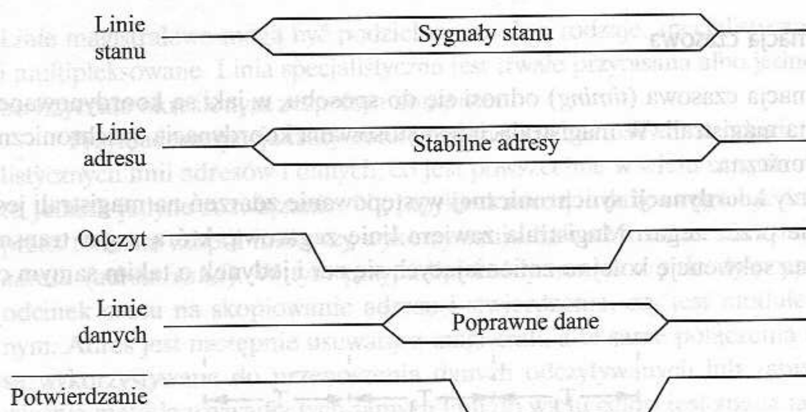
Koordynacja czasowa (*timing*) odnosi się do sposobu, w jaki są koordynowane zdarzenia na magistrali. W magistralach jest stosowana koordynacja synchroniczna lub asynchroniczna.

Przy **koordynacji synchronicznej** występowanie zdarzeń na magistrali jest wyznaczone przez zegar. Magistrala zawiera linię zegarową, którą zegar transmituje regularną sekwencję kolejno zmieniających się zer i jedynek o takim samym czasie

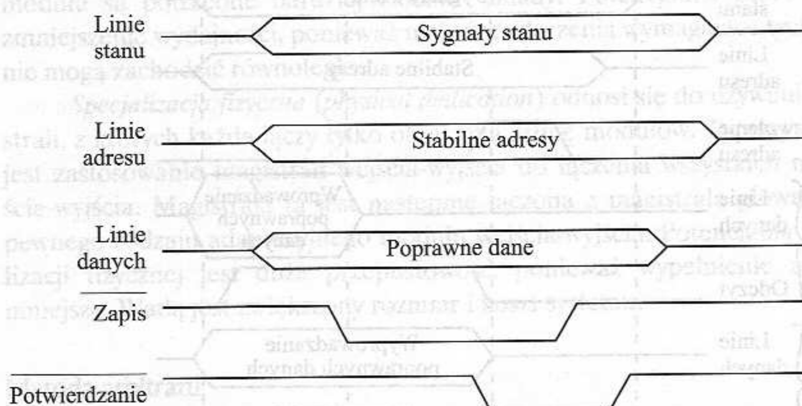


Rysunek 3.19. Przebiegi czasowe synchronicznych operacji na magistrali

trwania. Pojedyncza transmisja 1-0 jest nazywana *cyklem zegara* lub *cyklem magistrali* i określa przedział czasowy. Wszystkie inne urządzenia dołączone do magistrali mogą odczytywać stan linii zegarowej, a wszystkie zdarzenia rozpoczynają się równocześnie z cyklem zegara. Na rysunku 3.19a jest pokazany wykres czasowy dla synchronicznej operacji odczytu (zob. opis wykresów czasowych w dodatku 3A). Inne sygnały na magistrali mogą się zmieniać w momencie narastania sygnału zegarowego (z nieznacznym opóźnieniem wynikającym z czasu reakcji). Większość zdarzeń zajmuje tylko jeden cykl zegara. W tym prostym przykładzie procesor umieszcza adres na linii adresowej i może potwierdzać różne linie stanu. Gdy linie adresu zostaną ustabilizowane, procesor wysyła sygnał zezwolenia. W przypadku operacji odczytu procesor wysyła polecenie odczytu na początku drugiego cyklu. Moduł pamięci rozpoznaje adres oraz,



(a) Cykl odczytu magistrali systemowej



(b) Cykl zapisu magistrali systemowej

Rysunek 3.20. Przebiegi czasowe asynchronicznych operacji na magistrali

z opóźnieniem 1 cyklu, umieszcza dane w liniach danych. W przypadku operacji zapisu, procesor umieszcza dane na liniach danych na początku drugiego cyklu i wysyła polecenie zapisu po ustabilizowaniu linii danych. Moduł pamięci kopiuje informacje z linii danych podczas trzeciego cyklu zegara.

Przy **koordynacji asynchronicznej** występowanie zdarzeń na magistrali jest zależne od zdarzenia poprzedzającego. W prostym przykładzie odczytu z rys. 3.20a procesor umieszcza sygnały adresu i stanu na magistrali. Po pewnym czasie wymaganym do ustabilizowania sygnałów, wysyła polecenie odczytu, sygnalizując obecność ważnych sygnałów adresu i sterowania. Odpowiednia pamięć dekoduje adres i odpowiada, umieszczając dane na linii danych. Gdy linie danych się ustabilizują, moduł pamięci wysyła sygnał potwierdzenia w celu zasygnalizowania procesorowi, że dane są dostępne. Gdy urządzenie nadrzędne odczyta dane z linii danych, usuwa potwierdzenie sygnału odczytu. Sprawia to, że moduł pamięci opuszcza linie danych i potwierdzania. Na zakończenie, gdy linia potwierdzania zostanie opuszczona, urządzenie nadrzędne usuwa informacje adresowe.

Na rysunku 3.20b została pokazana prosta, asynchroniczna operacja zapisu. W tym przypadku urządzenie nadrzędne umieszcza dane na liniach danych w tym samym czasie, gdy umieszcza sygnały na liniach stanu i adresu. Moduł pamięci odpowiada na polecenie zapisu, kopiując dane z linii danych, następnie zaś umieszcza sygnał potwierdzenia na linii potwierdzania. Wówczas urządzenie nadrzędne porzuca sygnał zapisu, a moduł pamięci – sygnał potwierdzenia.

Koordynacja synchroniczna jest łatwiejsza do wdrożenia i testowania. Jest jednak mniej elastyczna niż asynchroniczna. Ponieważ wszystkie urządzenia dołączone do magistrali synchronicznej są zależne od ustalonej szybkości zegara, system nie może w pełni wykorzystać postępu w wydajności urządzeń. Przy koordynacji asynchronicznej magistrala może współpracować z urządzeniami szybkimi i wolnymi, wykorzystującymi nową i starą technologię.

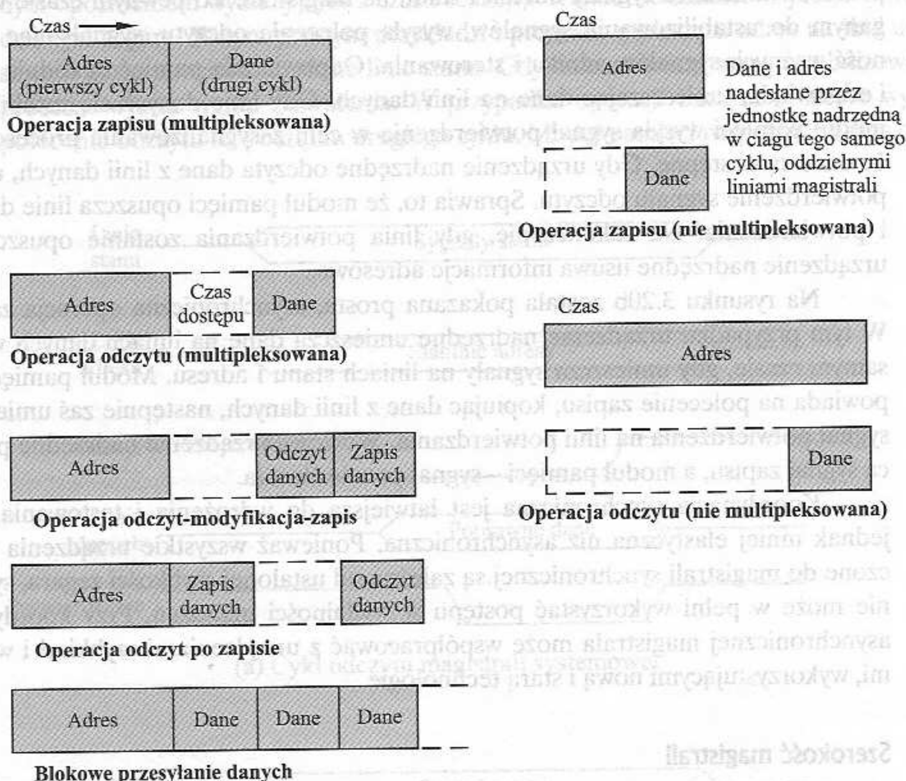
Szerokość magistrali

Wspomnieliśmy już o pojęciu szerokości magistrali. Szerokość szyny danych ma wpływ na wydajność systemu: im szersza jest szyna danych, tym większa jest liczba jednocześnie przesyłanych bitów. Szerokość szyny adresowej ma natomiast wpływ na pojemność systemu: im szersza jest szyna adresowa, tym większa jest ilość lokacji możliwych do określenia w pamięci.

Rodzaje transferu danych

Magistrala służy do przesyłania różnych rodzajów danych, co widać na rys. 3.21. Wszystkie magistrale obsługują zarówno zapis (transfer od modułu nadrzędnego do podrzędnego), jak i odczyt (transfer w przeciwnym kierunku). W przypadku multipleksowanych szyn adresów i danych szyna jest najpierw używana do specyfikowania adresu, a następnie do przesyłania danych. W operacji odczytu występuje typowe oczekiwanie, podczas gdy dane są pobierane z modułu podrzędnego

i wprowadzane do magistrali. Zarówno przy zapisie, jak i odczycie może również występować opóźnienie, jeśli zaistnieje potrzeba arbitrażu, aby przejąć sterowanie linią w celu wykonania pozostałej części operacji (np. w celu przejęcia magistrali przy żądaniu odczytu lub zapisu, a następnie ponownego przejęcia magistrali w celu wykonania odczytu lub zapisu).



Rysunek 3.21. Rodzaje magistralowego transferu danych [GOOR89]

W przypadku specjalistycznych szyn adresów i danych adres jest lokowany na szynie adresowej i pozostaje tam w czasie, gdy dane są doprowadzane do szyny danych. Przy operacji zapisu moduł nadrzędny umieszcza dane na szynie danych natychmiast po ustabilizowaniu się adresu, gdy moduł podrzędny ma możliwość rozpoznania adresu. Przy operacji odczytu moduł podrzędny umieszcza dane na szynie danych tuż po rozpoznaniu adresu i pobraniu danych.

Na niektórych magistralach są dopuszczalne operacje kombinowane. Operacja odczyt-modyfikacja-zapis jest po prostu odczytem, po którym natychmiast następuje zapis pod tym samym adresem. Adres jest rozgłaszany tylko raz, na początku operacji. W typowym przypadku cała operacja jest niepodzielna w celu zapobieżenia dostępowi do danych przez inne potencjalne moduły nadrzędne.

Zasadniczym celem tego rozwiązania jest ochrona zasobów pamięci w systemach wieloprogramowych (zob. rozdz. 8).

Odczyt po zapisie jest niepodzielną operacją składającą się z zapisu, po którym natychmiast następuje odczyt z tego samego adresu. Operacja odczytu może być wykonywana w celu kontroli.

Niektóre systemy magistralowe umożliwiają także blokowe przesyłanie danych. W tym przypadku po jednym cyklu adresu następuje *n* cykli danych. Pierwsza grupa danych jest przesyłana pod sprecyzowany adres (lub pobierana pod tym adresem), pozostałe dane są przesyłane pod następne adresy (lub pobierane pod tymi adresami).

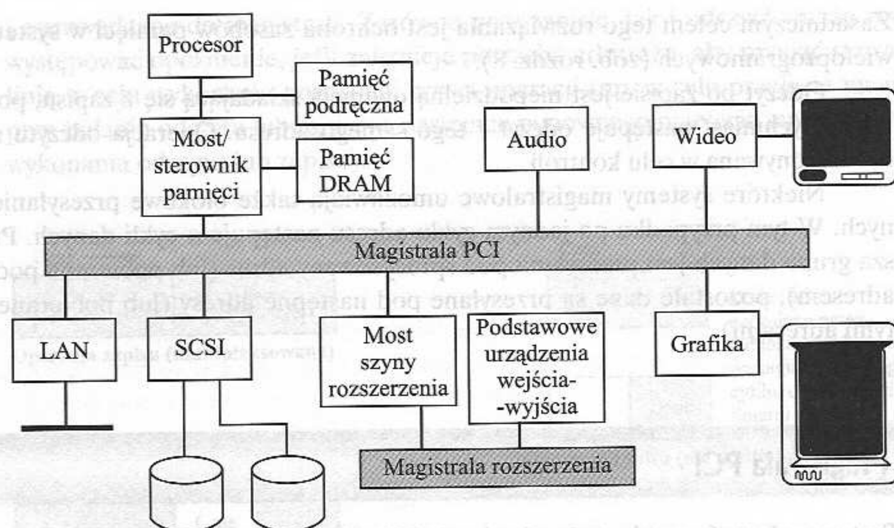
3.5. Magistrala PCI

System połączeń urządzeń peryferyjnych PCI (skrót od ang. *Peripheral Component Interconnect*) jest współczesną, szerokopasmową magistralą niezależną od procesora, która może funkcjonować jako magistrala „międzypiętrowa" (*mezzanine*) lub peryferyjna. W porównaniu z innymi, powszechnie spotykanymi magistralami PCI umożliwia uzyskanie większej wydajności systemu, jeśli są wykorzystywane szybkie podsystemy wejścia-wyjścia (np. urządzenia graficzne, sterowniki interfejsów sieciowych, sterowniki dysków i inne). Aktualna norma pozwala na użycie do 64 linii danych przy 66 MHz, co daje szybkość przesyłania danych 528 MB/s lub 4,224 Gbit/s. Jednak nie tylko duża szybkość stanowi o atrakcyjności PCI. Magistrala PCI została zaprojektowana jako ekonomiczne rozwiązanie spełniające wymagania wejścia-wyjścia w nowoczesnych systemach; wymaga niewielu mikroukładów i obsługuje działanie innych magistrali, które są z nią połączone.

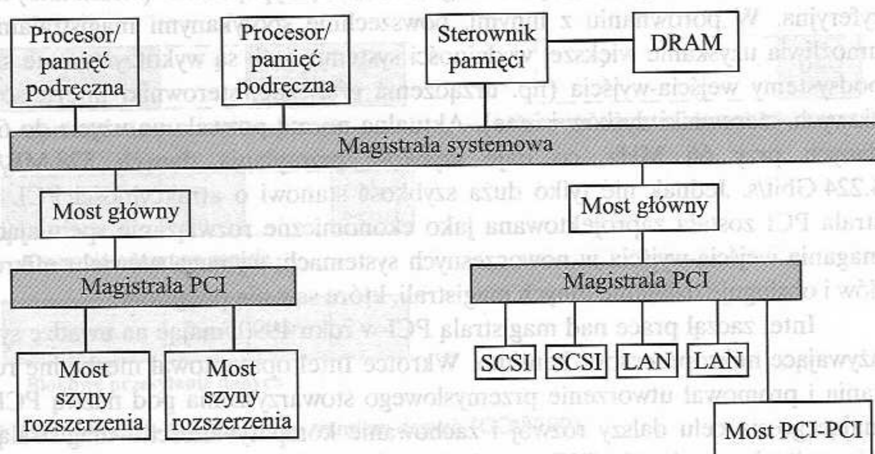
Intel zaczął prace nad magistralą PCI w roku 1990, mając na uwadze systemy używające mikroprocesora Pentium. Wkrótce Intel opatentował niezbędne rozwiązania i promował utworzenie przemysłowego stowarzyszenia pod nazwą PCI SIG, mającego na celu dalszy rozwój i zachowanie kompatybilności z magistralą PCI. W rezultacie magistrala PCI została szeroko zaakceptowana, a jej zastosowanie w komputerach osobistych, stacjonarnych i w systemach serwerów wzrasta. Wersją aktualną podczas pisania tej książki była PCI 2.2. Ponieważ specyfikacja jest dostępna publicznie i zyskała poparcie znacznej części przemysłu mikroprocesorów i urządzeń peryferyjnych, wyroby z magistralą PCI sprzedawane przez różnych dostawców są kompatybilne.

Magistralę PCI zaprojektowano dla szerokiego zakresu konfiguracji wykorzystujących mikroprocesory, łącznie z systemami jedno- i wieloprocesorowymi. Dlatego też PCI realizuje zespół funkcji o ogólnym przeznaczeniu. Wykorzystuje koordynację synchroniczną i arbitraż scentralizowany.

Na rysunku 3.22a jest pokazane typowe wykorzystanie magistrali PCI w systemie jednoprocesorowym. Zintegrowany układ sterownika DRAM i mostu do magistrali PCI zapewnia ścisłe sprzężenie z procesorem i możliwość dostarczania



(a) Typowy system biurkowy



(b) Typowy system serwera

Rysunek 3.22. Przykładowe konfiguracje magistrali PCI: (a) typowy system biurkowy; (b) typowy system serwera

danych z dużą szybkością. Most działa jako bufor danych, dzięki czemu szybkość magistrali PCI może się różnić od szybkości procesora. W systemie wieloprocesorowym (rys. 3.22b) jedną lub więcej konfiguracji magistrali PCI można połączyć za pomocą mostów z magistralą systemową procesorów. Magistrala systemowa obsługuje jedynie zespoły procesor/pamięć podręczna, pamięć główną i mosty PCI. Również i tutaj zastosowanie mostów uniezależnia magistralę PCI od szybkości procesora, umożliwiając jednocześnie szybkie otrzymywanie i dostarczanie danych.

Struktura magistrali

Magistrala PCI może być konfigurowana jako magistrala 32- lub 64-bitowa. W tabeli 3.3 są przedstawione obowiązujące linie sygnałowe magistrali PCI. Jest ich 49. Można je podzielić na następujące grupy funkcjonalne:

- **Wyprowadzenia systemowe.** Należą do nich wyprowadzenia zegara i przywracania (*reset*).
- **Wyprowadzenia adresu i danych.** Należą do nich 32 linie multipleksowane adresów i danych. Pozostałe linie w tej grupie są wykorzystywane do interpretowania i określania ważności linii sygnałowych przenoszących adresy i dane.
- **Wyprowadzenia sterowania interfejsu.** Służą do koordynowania transakcji i współdziałania między jednostkami inicjującymi a docelowymi.
- **Wyprowadzenia arbitrażowe.** W przeciwieństwie do pozostałych linii sygnałowych magistrali PCI nie są one liniami wspólnymi. Każda jednostka nadrzędna PCI ma własną parę linii arbitrażowych, które łączą ją bezpośrednio z arbitrem magistrali PCI.
- **Wyprowadzenia informujące o błędach.** Wykorzystywane do informowania o błędach parzystości i innych.

Ponadto specyfikacja PCI określa 51 opcjonalnych linii sygnałowych (tabela 3.4), podzielonych na następujące grupy:

- **Wyprowadzenia przerwania.** Są one przewidziane dla urządzeń PCI, które muszą generować zapotrzebowanie na obsługę. Tak jak w przypadku wyprowadzeń arbitrażowych, nie są one wspólne. Każde urządzenie PCI dysponuje własną linią lub liniami przerwań łączącymi ze sterownikiem przerwań.
- **Wyprowadzenia obsługi pamięci podręcznej.** Są one wykorzystywane do obsługi pamięci podręcznych zawartych w procesorach i innych urządzeniach PCI. Umożliwiają stosowanie protokołu podglądania (*snoopy*) pamięci podręcznej. Protokoły te są omówione w rozdz. 18.
- **Wyprowadzenia rozszerzenia magistrali 64-bitowej.** Należą do nich 32 linie multipleksowane adresów i danych. W połączeniu z obowiązującymi liniami adresów i danych tworzą 64-bitową magistralę adresów i danych. Inne linie z tej grupy są używane do interpretowania i określania ważności linii sygnałowych przenoszących dane i adresy. Wreszcie występują tu dwie linie umożliwiające dwóm urządzeniom PCI uzgodnienie użycia szerokości 64-bitowej.
- **Wyprowadzenia testowania granic JTAG.** Linie te umożliwiają realizację procedur testowania określonych w normie IEEE 1149.1.

Tabela 3.3. Obowiązujące linie sygnałowe standardu PCI

Oznaczenie	Rodzaj	Opis
		Wyprowadzenia systemowe
CLK	we	Zapewnia koordynację czasową wszystkich transakcji i jest próbkowana przez wszystkie wejścia podczas zbocza rosnącego. Częstotliwość zegara do 66 MHz.
RST#	we	Wymusza inicjowanie wszystkich zgodnych ze standardem PCI rejestrów, sekwenserów i sygnałów.
		Wyprowadzenia adresów i danych
AD[31÷0]	t/s	Multipleksowane linie do przenoszenia adresów i danych.
C/BE[3÷0]#	t/s	Multipleksowane sygnały rozkazów magistralowych oraz zezwolenia bajtów. Podczas fazy danych linie te wskazują, które z czterech szyn bajtów niosą znaczące dane.
PAR	t/s	Dostarcza sygnały parzystości linii AD i C/BE z opóźnieniem jednego cyklu zegara. Jednostka nadrzędna aktywuje linie PAR w fazach adresu i zapisu danych; jednostka docelowa aktywuje linie PAR w fazach odczytu danych.
		Wyprowadzenia sterowania interfejsem
FRAME#	s/t/s	Aktywowana przez bieżącą jednostkę nadrzędną w celu wskazania początku i czasu trwania transakcji. Jest potwierdzana na początku i negowana, gdy inicjator jest gotowy do rozpoczęcia końcowej fazy danych.
IRDY#	s/t/s	Gotowość inicjatora. Aktywowana przez bieżącą jednostkę nadrzędną magistrali (inicjatora transakcji). Podczas odczytu wskazuje, że jednostka nadrzędna jest gotowa do przyjęcia danych; podczas zapisu wskazuje, że na AD obecne są ważne dane.
TRDY#	s/t/s	Gotowość jednostki docelowej. Aktywowana przez jednostkę docelową (wybrany przyrząd). Podczas odczytu wskazuje, że na AD są obecne ważne dane; podczas zapisu wskazuje, że jednostka docelowa jest gotowa do przyjęcia danych.
STOP#	s/t/s	Wskazuje, że bieżąca jednostka docelowa domaga się od inicjatora zatrzymania bieżącej transakcji.
IDSEL	we	Wybór przyrządu inicjowania. Używana do wyboru mikroukładu podczas konfigurowania transakcji odczytu i zapisu.
DEVSEL#	we	Wybór przyrządu. Potwierdzana przez jednostkę docelową, gdy rozpoznała ona swój adres. Wskazuje bieżącemu inicjatorowi, czy został wybrany jakikolwiek przyrząd.
		Wyprowadzenia arbitrażowe
REQ#	t/s	Wskazuje arbitrowi, że ten przyrząd wymaga użycia magistrali. Jest to linia doprowadzona do konkretnego przyrządu.
GNT#	t/s	Wskazuje przyrządowi, że arbiter przekazał mu dostęp do magistrali. Jest to linia doprowadzona do konkretnego przyrządu.
		Wyprowadzenia informowania o błędzie
PERR#	s/t/s	Błąd parzystości. Wskazuje, że został wykryty błąd parzystości danych przez jednostkę docelową podczas fazy zapisu danych lub przez inicjatora podczas fazy odczytu danych.
SERR#	o/d	Błąd systemowy. Może być wzbudzona przez jakikolwiek przyrząd w celu poinformowania o błędach parzystości adresu oraz o błędach krytycznych innych niż błędy parzystości.

Tabela 3.4. Opcjonalne linie sygnałowe standardu PCI

Oznaczenie	Rodzaj	Opis
		Doprowadzenia przerwania
INTA#	o/d	Używana do zgłaszania zapotrzebowania na przerwanie.
INTB#	o/d	Używana do zgłaszania zapotrzebowania na przerwanie; ma znaczenie tylko w odniesieniu do przyrządu wielofunkcyjnego.
INTC#	o/d	Używana do zgłaszania zapotrzebowania na przerwanie; ma znaczenie tylko w odniesieniu do przyrządu wielofunkcyjnego.
INTD#	o/d	Używana do zgłaszania zapotrzebowania na przerwanie; ma znaczenie tylko w odniesieniu do przyrządu wielofunkcyjnego.
		Doprowadzenia wspierające pamięć podręczną
SBO#	we-wy	Wycofanie się z podglądania. Wskazuje na trafienie zmodyfikowanego wiersza.
SDONE	we-wy	Zakończenie podglądania. Wskazuje stan podglądania. Potwierdzana po zakończeniu podglądania.
		Wyprowadzenia rozszerzenia magistrali do 64 bitów
AD[63÷32]	t/s	Linie multipleksowane używane do adresów i danych w celu rozszerzenia magistrali do 64 bitów.
C/BE[7÷4]#	t/s	Multipleksowane sygnały rozkazów magistralowych i sygnałów zezwolenia bajtów. Podczas fazy adresu linie te dostarczają dodatkowych rozkazów magistralowych. Podczas fazy danych linie te wskazują, która z czterech rozszerzonych szyn bajtowych przenosi znaczące dane.
REQ64#	s/t/s	Używana do zgłaszania zapotrzebowania na transfer w pakietach po 64 bity.
ACK64#	s/t/s	Wskazuje, że jednostka docelowa chce dokonać transferu 64-bitowego.
PAR64	t/s	Dostarcza sygnałów parzystości rozszerzonych linii AD i C/BE z opóźnieniem jednego cyklu zegara.
		Wyprowadzenia testowania granic JTAG
TCK	we	Zegar testowy. Używana do synchronizowania wprowadzania i wyprowadzania informacji o stanie i danych testowych do (i z) przyrządu podczas skanowania granic.
TDI	we	Wejście testowe. Używana do szeregowego wprowadzania danych testowych i rozkazów do przyrządu.
TDO	wy	Wyjście testowe. Używana do szeregowego wyprowadzania danych testowych i rozkazów z przyrządu.
TMS	we	Wybór trybu testowania. Używana do kontrolowania stanu sterownika portu testowego.
TRST#	we	Inicjowanie testu. Używana do inicjowania sterownika portu testowego.

we – tylko sygnał wejściowy,
wy – tylko sygnał wyjściowy,
t/s – dwukierunkowy, trójstanowy sygnał we-wy,
s/t/s – podtrzymywany sygnał trójstanowy, aktywowany w danej chwili tylko przez jeden przyrząd,
o/d – otwarty dren: pozwala wielu przyrządom na wspólne używanie w formie układowego OR (LUB),
– aktywny stan sygnału występuje przy niskim napięciu.

Rozkazy PCI

Działanie magistrali odbywa się w formie transakcji między inicjatorem (modułem nadrzędnym) a celem (modułem podrzędnym). Gdy inicjator domaga się sterowania magistralą, określa typ transakcji, która ma być przeprowadzona. Podczas fazy

adresu transakcji do sygnalizowania typu transakcji są używane linie C/BE. Rozkazy są następujące:

- potwierdzenie przerwania,
- cykl specjalny,
- odczyt wejścia-wyjścia,
- zapis wejścia-wyjścia,
- odczyt pamięci,
- linia odczytu pamięci,
- zwielokrotniony odczyt pamięci,
- zapis w pamięci,
- zapis w pamięci i unieważnienie,
- odczyt konfiguracji,
- zapis konfiguracji,
- cykl podwójnego adresu.

„Potwierdzenie przerwania" jest rozkazem odczytu przeznaczonym dla urządzenia, które działa jako sterownik przerwań na magistrali PCI. Linie adresowe nie są używane podczas fazy adresu, a linie zezwolenia bajta wskazują rozmiar identyfikatora przerwania, który musi być zwrócony.

„Cykl specjalny" jest rozkazem używanym przez inicjatora w celu rozgłoszenia wiadomości przeznaczonych dla jednego lub więcej celów.

Rozkazy „odczyt wejścia-wyjścia" i „zapis wejścia-wyjścia" są używane do przenoszenia danych między inicjatorem a sterownikiem wejścia-wyjścia. Każde urządzenie wejścia-wyjścia ma własną przestrzeń adresową. Linie adresowe służą do wskazywania określonego urządzenia oraz do danych, które mają być wymienione z tym urządzeniem. Koncepcją adresów wejścia-wyjścia zajmiemy się w rozdz. 7.

Rozkazy odczytu i zapisu w pamięci są stosowane do specyfikowania przesyłania pakietu danych (*burst of data*) trwającego przez jeden lub więcej cykli zegarowych. Interpretacja tych rozkazów zależy od tego, czy sterownik pamięci na magistrali PCI realizuje protokół PCI przeznaczony do przesyłania danych między pamięcią główną a pamięcią podręczną. Jeśli tak, to przesyłanie danych do (i z) pamięci odbywa się typowo w postaci wierszy lub bloków[2]. Trzy rozkazy odczytu pamięci mają zastosowania wymienione w tabeli 3.5. Rozkaz „zapisz w pamięci" jest stosowany do przenoszenia danych do pamięci, w czasie jednego lub wielu cykli danych. Rozkaz „zapis w pamięci i unieważnienie" również służy do przenoszenia danych do pamięci, w czasie jednego lub wielu cykli. Ponadto gwarantuje on, że przynajmniej jedna linia pamięci podręcznej jest zapisana. Rozkaz ten obsługuje funkcję pamięci podręcznej polegającą na opóźnionym zapisie wiersza w pamięci.

[2] Podstawowe zasady dotyczące pamięci podręcznych są opisane w rozdziale 4; protokoły pamięci podręcznych w systemach magistralowych są opisane w rozdziale 18.

Tabela 3.5. Interpretacja rozkazów odczytu standardu PCI

Rodzaj rozkazu odczytu	Pamięć współpracująca z pamięcią podręczną	Pamięć główna bez pamięci podręcznej
Odczyt z pamięci	pakietowanie co najwyżej 1/2 wiersza pamięci podręcznej	pakietowanie co najwyżej 2 cykli transferu
Odczyt wiersza z pamięci	pakietowanie od 1/2 do 3 wierszy pamięci podręcznej	pakietowanie od 3 do 12 cykli transferu danych
Wielokrotny odczyt z pamięci	pakietowanie ponad 3 wierszy pamięci podręcznej	pakietowanie ponad 12 cykli transferu danych

Dwa rozkazy konfiguracyjne umożliwiają inicjatorowi odczytanie i zaktualizowanie parametrów konfiguracji w urządzeniu dołączonym do magistrali PCI. Każde urządzenie PCI może zawierać do 256 rejestrów wewnętrznych, które są wykorzystywane do konfigurowania tego urządzenia podczas inicjowania systemu.

Rozkaz „cykl podwójnego adresu" jest wykorzystywany przez inicjatora do wskazania, że stosuje on adresowanie 64-bitowe.

Przesyłanie danych

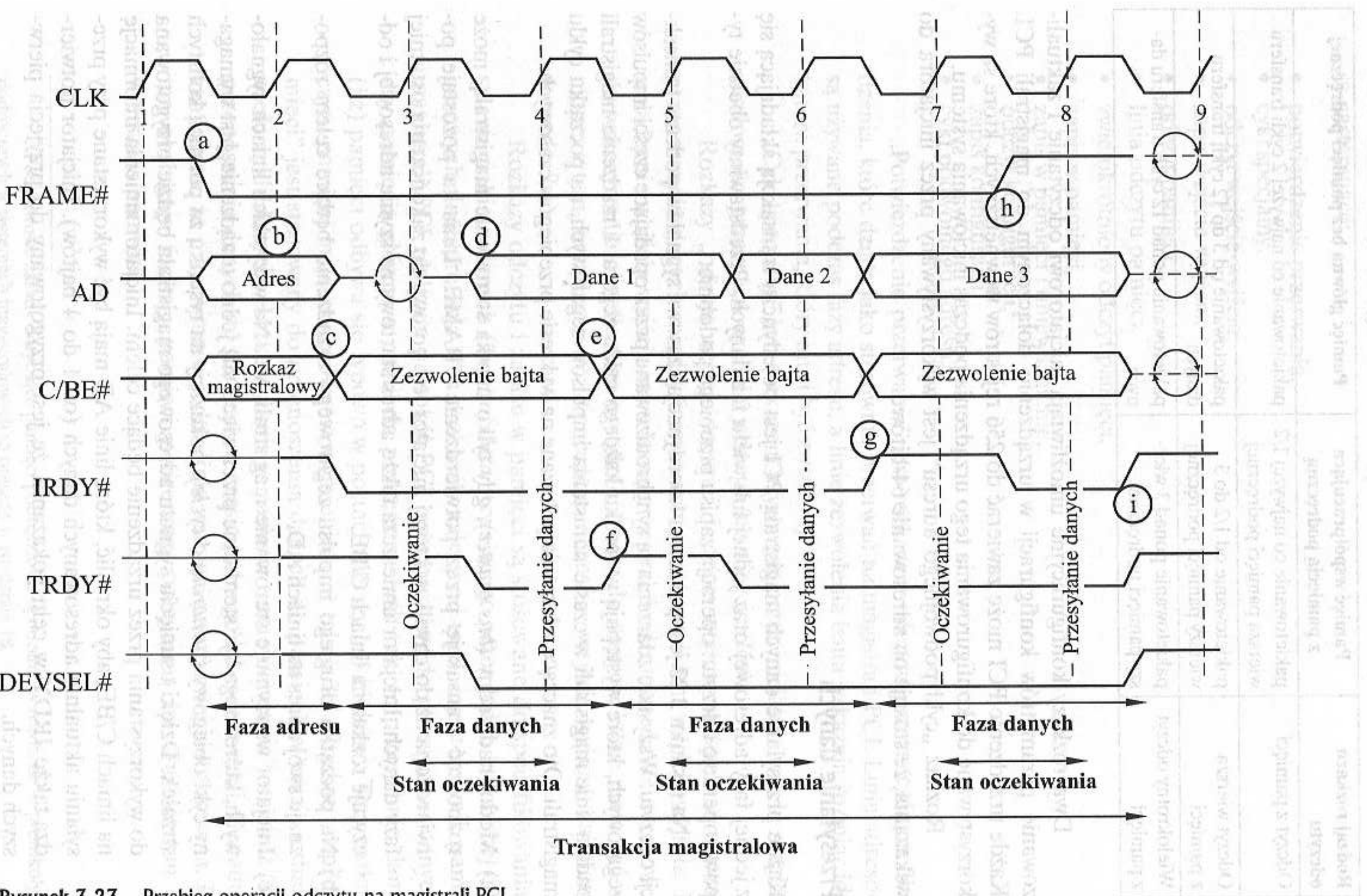
Każde przesyłanie danych magistralą PCI jest pojedynczą transakcją składającą się z jednej fazy adresowej oraz jednej lub wielu faz danych. Przedstawimy obecnie typową operację odczytu; operacja zapisu przebiega podobnie.

Na rysunku 3.23 jest pokazana koordynacja czasowa sygnałów podczas transakcji odczytu. Wszystkie zdarzenia są synchronizowane przez opadające części impulsów zegarowych, które występują w środku każdego cyklu zegara. Urządzenia magistrali badają linie magistrali w czasie narastania impulsów zegarowych, na początku cyklu magistrali. Oto znaczące zdarzenia pokazane na wykresie przebiegów czasowych:

(a) Moduł nadrzędny (*bus master*), gdy tylko uzyska sterowanie magistralą, może rozpocząć transakcję przez potwierdzenie FRAME. Linia ta pozostaje potwierdzona aż do chwili, w której inicjator jest gotowy do zakończenia ostatniej fazy danych. Inicjator umieszcza także adres startowy na szynie adresowej i odczytuje rozkaz na liniach C/BE.

(b) Na początku drugiego impulsu zegarowego urządzenie będące celem rozpoznaje swój adres na liniach AD.

(c) Inicjator wstrzymuje sterowanie magistrali AD. Na wszystkich liniach sygnałowych, które mogą być sterowane przez więcej niż jedno urządzenie, jest wymagany cykl obiegowy (*turnaround cycle*), pokazany na rysunku za pomocą kolistych strzałek. Dzięki usunięciu sygnału adresowego magistrala będzie przygotowana do wykorzystania przez urządzenie będące celem. Inicjator zmienia informację na liniach C/BE, aby określić, które linie AD mają być wykorzystane przy przesyłaniu aktualnie adresowanych danych (od 1 do 4 bajtów). Inicjator potwierdza także IRDY w celu pokazania, że jest przygotowany do przyjęcia pierwszych danych.



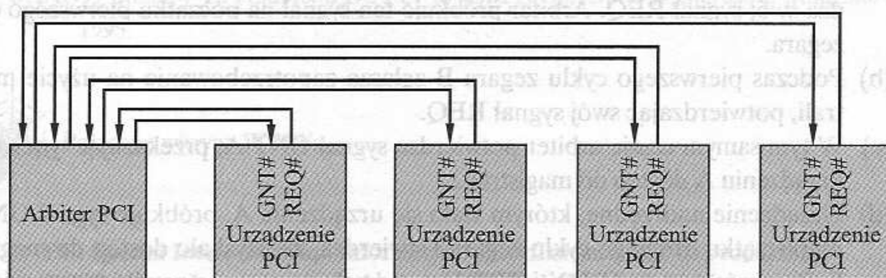
Rysunek 3.23. Przebieg operacji odczytu na magistrali PCI

(d) Wybrany cel potwierdza DEVSEL, aby pokazać, że rozpoznał swój adres i udzieli odpowiedzi. Umieszcza wymagane dane na liniach AD i potwierdza TRDY, aby pokazać, że na magistrali są obecne ważne dane.
(e) Inicjator odczytuje dane na początku czwartego impulsu zegarowego i zmienia stan linii zezwolenia bajta (*byte enable lines*) w celu przygotowania do następnego odczytu.
(f) W omawianym przykładzie cel potrzebuje pewnego czasu, aby przygotować się do transmisji drugiego bloku danych. Usuwa więc potwierdzenie TRDY, aby zasygnalizować inicjatorowi, że w ciągu nadchodzącego cyklu nie będzie nowych danych. Zgodnie z tym inicjator nie odczytuje linii danych na początku piątego cyklu zegara i nie musi zmieniać stanu linii zezwolenia bajta podczas tego cyklu. Blok danych jest odczytywany na początku szóstego cyklu.
(g) Podczas szóstego cyklu zegara cel umieszcza trzecią porcję danych na magistrali. Jednak w tym przykładzie inicjator nie jest jeszcze gotowy do odczytywania porcji danych (może być w stanie czasowo zapełnionych buforów). Odwołuje więc potwierdzenie IRDY, co powoduje, że cel będzie zachowywał trzecią porcję danych na magistrali w ciągu następnego cyklu zegara.
(h) Inicjator wie, że trzecie przesłanie danych będzie ostatnim, odwołuje więc FRAME, aby w ten sposób zasygnalizować celowi, że jest to właśnie ostatnie przesłanie. Potwierdza także IRDY w celu pokazania, że jest gotów do zakończenia tego przesłania.
(i) Inicjator odwołuje IRDY, powodując powrót magistrali do stanu jałowego, natomiast cel odwołuje TRDY i DEVSEL.

Arbitraż

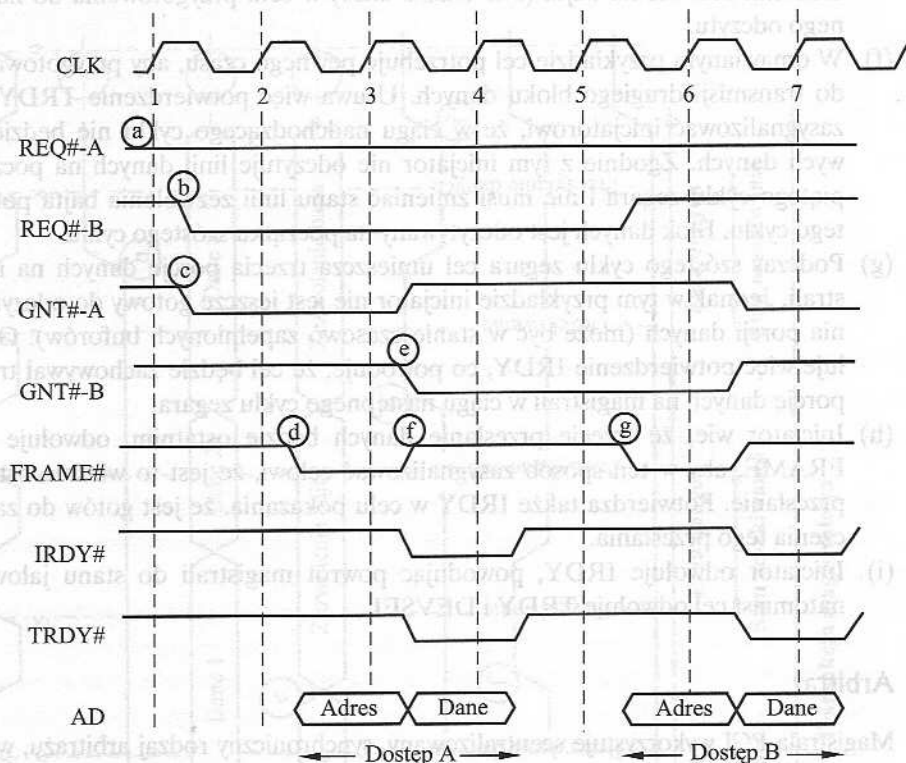
Magistrala PCI wykorzystuje scentralizowany, synchroniczny rodzaj arbitrażu, w którym każdy moduł nadrzędny ma przypisane unikatowe sygnały zapotrzebowania (REQ) i udostępniania (*grant* – GNT). Odpowiednie linie sygnałowe są podłączone do centralnego arbitra (rys. 3.24), a dostęp do magistrali jest uzyskiwany przez zgodne zgłoszenie zapotrzebowania i udostępniania.

Specyfikacja PCI nie dyktuje szczególnego algorytmu arbitrażu. Arbiter może wykorzystywać podejście „pierwszy zgłoszony – pierwszy obsłużony", rozwiązanie cy-



Rysunek 3.24. Arbiter magistrali PCI

cykliczne (*round-robin*) lub jakiś rodzaj układu priorytetów. Moduł nadrzędny PCI musi korzystać z arbitrażu przy każdej transakcji, którą chce przeprowadzić, ponieważ pojedyncza transakcja składa się z fazy adresu, po której następuje jedna lub wiele faz danych.



Rysunek 3.25. Arbitraż magistrali PCI między dwiema jednostkami nadrzędnymi

Na rysunku 3.25 jest pokazany przykład, w którym urządzenia A i B odwołują się do arbitrażu. Realizowana jest następująca sekwencja zdarzeń:

(a) W pewnym momencie, przed rozpoczęciem pierwszego cyklu zegara, A potwierdza swój sygnał REQ. Arbiter próbkuje ten sygnał na początku pierwszego cyklu zegara.

(b) Podczas pierwszego cyklu zegara B zgłasza zapotrzebowanie na użycie magistrali, potwierdzając swój sygnał REQ.

(c) W tym samym czasie arbiter potwierdza sygnał GNT-A, przekazując tym samym urządzeniu A dostęp do magistrali.

(d) Urządzenie nadrzędne, którym stało się urządzenie A, próbkuje sygnał GNT-A na początku drugiego cyklu zegara i stwierdza, że uzyskało dostęp do magistrali. Zauważa też, że IRDY i TRDY są odwołane, z czego wynika, że magistrala jest wolna. Potwierdza więc FRAME, umieszcza informację adresową na szynie

adresu i rozkaz na magistrali C/BE (nie pokazane). Kontynuuje także potwierdzanie REQ-A, ponieważ ma jeszcze drugą transakcję do przeprowadzenia po pierwszej.

(e) Arbiter magistrali próbkuje wszystkie linie REQ na początku trzeciego cyklu zegara i decyduje, że udostępnia magistralę urządzeniu B w celu dokonania następnej transakcji. Potwierdza więc GNT-B i odwołuje GNT-A. Urządzenie B nie będzie mogło użyć magistrali, dopóki nie powróci ona do stanu jałowego.

(f) A odwołuje FRAME w celu pokazania, że realizowane jest ostatnie (i jedyne) przesłanie. Umieszcza dane na magistrali danych i sygnalizuje to celowi za pomocą IRDY. Cel odczytuje dane na początku następnego cyklu zegara.

(g) Na początku piątego cyklu zegara B stwierdza odwołanie IRDY i FRAME, może więc przejąć sterowanie magistralą, potwierdzając FRAME. Odwołuje także swój sygnał REQ, ponieważ chce przeprowadzić tylko jedną transakcję.

W rezultacie A uzyskuje dostęp do magistrali i może przeprowadzić swoją drugą transakcję.

Zauważmy, że arbitraż może mieć miejsce w tym samym czasie, w którym aktualna jednostka nadrzędna linii przesyła dane. Dzięki temu arbitraż nie powoduje utraty cyklu magistrali. Takie rozwiązanie jest określane jako *arbitraż ukryty* (*hidden arbitration*).

3.6. Polecana literatura i witryny WWW

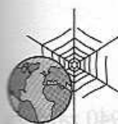
Literatura na temat magistral i innych struktur połączeń jest zaskakująco skromna. W [ALEX93] zawarto głębsze omówienie struktur magistralowych i problemów przesyłania za pomocą magistrali, uwzględniając kilka szczególnych przypadków magistrali.

Najbardziej klarowny opis magistrali PCI zawarto w [SHAN95]. Także [ABBO00] zawiera wiele solidnych informacji o PCI.

ABBO00 Abbot D.: *PCI Bus Demistified.* Eagle Rock, LLH Technology Publishing, 2000.

ALEX93 Alexandridis N.: *Design of Microprocessor-Based Systems*. Englewood Cliffs, Prentice Hall, 1993.

SHAN95 Shanley T., Anderson D.: *PCI Systems Architecture*. Richardson, Mindshare Press, 1995.



Polecane witryny WWW:

- **PCI Special Interest Group.** Informacje o specyfikacjach PCI i o produktach.
- **PCI Pointers.** Łącza do witryn dostawców PCI i do innych źródeł informacji.

3.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Arbitraż centralizowany – *centralized arbitration*
Arbitraż magistralowy – *bus arbitration*
Arbitraż rozproszony – *distributed arbitration*
Cykl rozkazu – *cycle instruction*
Koordynacja asynchroniczna – *asynchronous timing*
Koordynacja synchroniczna – *synchronous timing*
Magistrala – *bus*
Magistrala PCI – *peripheral component interconnect* (PCI)
Magistrala systemowa – *system bus*
Pobieranie rozkazu – *instruction fetch*
Podprogram obsługi przerwań – *interrupt service routine*
Program obsługi przerwań – *interrupt handler*
Przerwanie – *interrupt*
Przerwanie zablokowane – *disable interrupt*
Rejestr adresowy pamięci (MAR) – *memory address register* (MAR)
Rejestr buforowy pamięci (MBR) – *memory buffer register* (MBR)
Szerokość magistrali – *bus width*
Szyna adresowa – *address bus*
Wykonywanie rozkazu – *instruction execute*

Pytania kontrolne

3.1. Jakie ogólne kategorie funkcji są specyfikowane w rozkazach komputerów?
3.2. Wymień i krótko zdefiniuj możliwe stany określające wykonywanie rozkazów.
3.3. Wymień i krótko zdefiniuj dwa podejścia do wielokrotnych przerwań.
3.4. Jakie rodzaje przesyłania danych musi obsługiwać struktura połączeń komputera (np. magistrala)?
3.5. Jakie są korzyści wynikające ze stosowania architektury z wieloma magistralami w porównaniu z architekturą opartą na jednej magistrali?
3.6. Wymień i krótko zdefiniuj grupy funkcjonalne linii sygnałowych w magistrali PCI.

Problemy do rozwiązania

3.1. Hipotetyczna maszyna z rys. 3.4 ma również dwa rozkazy wejścia-wyjścia:

0011 = Ładuj akumulator z wejścia-wyjścia
0111 = Przenieś zawartość akumulatora do wejścia-wyjścia

W tych przypadkach 12-bitowy adres identyfikuje urządzenie wejścia-wyjścia. Wykorzystując format z rys. 3.5, pokaż realizację poniższego programu:

1. Ładuj akumulator z urządzenia 5.
2. Dodaj zawartość pamięci z pozycji 940.
3. Zawartość akumulatora przenieś do urządzenia 6.

Przyjmij, że następną wartością pobieraną z urządzenia 5 jest 3 i że pozycja 940 zawiera wartość 2.

3.2. Wykonywanie programu przedstawionego na rys. 3.5 jest opisane w tekście przy użyciu sześciu kroków. Poszerz ten opis w celu zademonstrowania użycia MAR i MBR.

3.3. Rozważ hipotetyczny mikroprocesor 32-bitowy mający 32-bitowe rozkazy złożone z dwóch pól: pierwszy bajt zawiera kod operacji, pozostałość zaś argument natychmiastowy lub adres argumentu.

(a) Jaka jest maksymalna pojemność pamięci bezpośrednio adresowalnej (w bajtach)?

(b) Przeanalizuj wpływ na szybkość systemu, jeśli magistrala mikroprocesora ma:
1. 32-bitową lokalną szynę adresu oraz 16-bitową lokalną szynę danych lub
2. 16-bitową lokalną szynę adresu oraz 16-bitową lokalną szynę danych.

(c) Ile bitów musi zawierać licznik programu i rejestr rozkazu?

Źródło: [ALEX93].

3.4. Rozważ hipotetyczny mikroprocesor generujący adresy 16-bitowe (załóżmy np., że licznik programu i rejestry adresu są 16-bitowe) i dysponujący 16-bitową szyną danych.

(a) Jaka jest maksymalna przestrzeń adresowa pamięci, do której procesor może mieć bezpośredni dostęp, jeśli jest połączony z pamięcią 16-bitową?

(b) Jaka jest maksymalna przestrzeń adresowa pamięci, do której procesor może mieć bezpośredni dostęp, jeśli jest połączony z pamięcią 8-bitową?

(c) Jaka cecha architektury pozwoli temu mikroprocesorowi na dostęp do oddzielnej przestrzeni wejścia-wyjścia?

(d) Jeśli rozkaz wejścia i wyjścia może precyzować adres portu wejścia-wyjścia, ile 8-bitowych portów wejścia-wyjścia może obsługiwać mikroprocesor, a ile portów 16-bitowych? Uzasadnij odpowiedź.

Źródło: [ALEX93].

3.5. Rozważ mikroprocesor 32-bitowy z 16-bitową zewnętrzną szyną danych, sterowany zegarem wejściowym 8 MHz. Załóżmy, że mikroprocesor ma cykl magistrali, którego maksymalny czas trwania jest równy 4 cyklom zegara wejściowego. Jaka jest maksymalna szybkość transferu danych, z którą ten mikroprocesor może pracować? Aby zwiększyć wydajność, czy byłoby lepiej zastosować 32-bitową szynę danych, czy też podwoić częstotliwość zewnętrznego zegara dołączonego do mikroprocesora? Sprecyzuj inne możliwe założenia i objaśnij je.

Źródło: [ALEX93].

3.6. Rozważ system komputerowy, który zawiera moduł wejścia-wyjścia sterujący prostym dalekopisem z klawiaturą. W CPU są zawarte następujące rejestry, przyłączone bezpośrednio do magistrali systemowej:

INPR: Rejestr wejściowy – 8 bitów,
OUTR: Rejestr wyjściowy – 8 bitów,
FGI: Znacznik wejściowy – 1 bit,
FGO: Znacznik wyjściowy – 1 bit,
IEN: Zezwolenie przerwania – 1 bit.

Wejście z klawiatury dalekopisu i wyjście do drukarki dalekopisu są sterowane przez moduł wejścia-wyjścia. Dalekopis może kodować symbole alfanumeryczne w postaci słów 8-bitowych oraz dekodować słowa 8-bitowe na symbole alfanumeryczne.

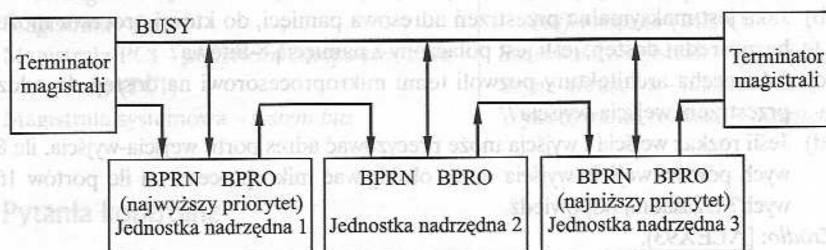
(a) Opisz, jak procesor może uzyskać dostęp do wejścia-wyjścia za pomocą dalekopisu, posługując się pierwszymi czterema z wymienionych rejestrów.

(b) Opisz, jak można tę funkcję zrealizować efektywniej, wykorzystując także IEN.

3.7. Na rysunku 3.26 widać schemat układu arbitrażu rozproszonego, który może być stosowany z przestarzałym już rozwiązaniem magistrali Multibus I. Moduły są fizycznie połączone łańcuchowo, według kolejności priorytetów. Moduł z lewej strony rysunku otrzy-

muje stały priorytet na magistrali, w postaci sygnału BPRN wskazującego, że żaden z modułów o wyższym priorytecie nie domaga się kontroli nad magistralą. Jeśli moduł nie żąda dostępu do magistrali, potwierdza swoją *linię wyłączenia priorytetu*, BPRO. Na początku cyklu zegara dowolny moduł może życzyć sobie sterowania magistralą przez ustalenie niskiego stanu linii BPRO. Obniża to stan linii BPRN następnego modułu w łańcuchu, przez co z kolei jest on zmuszany do obniżenia stanu swojej linii BPRO. Sygnał rozchodzi się więc wzdłuż łańcucha. Na zakończenie tej „reakcji łańcuchowej" powinien być tylko jeden moduł, którego linia BPRN jest potwierdzona, a którego linia BPRO jest w stanie niskim. Ten właśnie moduł ma priorytet. Jeśli na początku cyklu magistrali nie jest ona zajęta (linia BUSY nieaktywna), to moduł mający priorytet może przejąć kontrolę nad magistralą, potwierdzając BUSY.

Propagacja sygnału BPR od modułu o najwyższym priorytecie do modułu o najniższym priorytecie zabiera pewien czas. Wyjaśnij, czy ten czas musi być krótszy od cyklu zegara.



Rysunek 3.26. Arbitraż rozproszony Multibus I

3.8. W magistralach VAX SBI użyto rozproszonego arbitrażu synchronicznego. Każde urządzenie SBI (np. procesor, pamięć, moduł wejścia-wyjścia) ma jednoznaczny priorytet i jest przypisane do unikatowej linii zapotrzebowania na transfer (TR). SBI ma 16 takich linii (TR0, TR1, ..., TR15), przy czym TR0 ma najwyższy priorytet. Gdy urządzenie chce wykorzystać magistralę, rezerwuje przyszły przedział czasu (*time slot*) przez potwierdzenie swojej linii TR podczas bieżącego przedziału czasowego. Przy końcu bieżącego przedziału czasowego, każde urządzenie o zawieszonej rezerwacji bada linie TR; urządzenie o najwyższym priorytecie, mające rezerwację, wykorzysta najbliższy przedział czasowy.

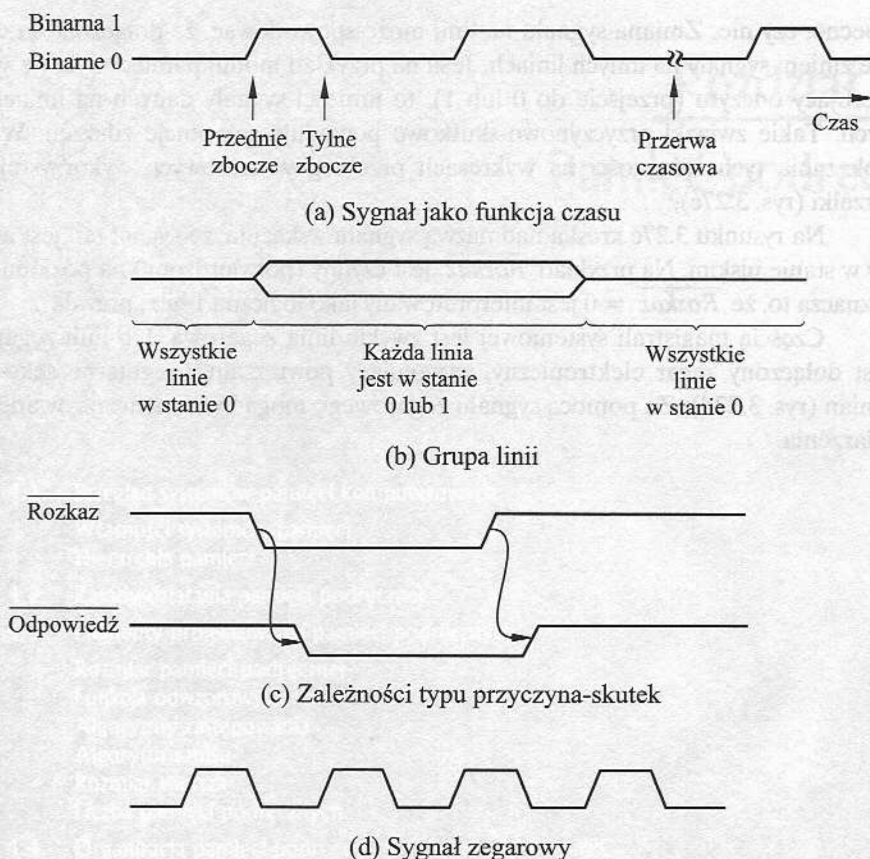
Do magistrali można dołączyć maksymalnie 17 urządzeń. Urządzenie o priorytecie 16 nie ma linii TR. Dlaczego?

3.9. Paradoksalnie, urządzenie o najniższym priorytecie ma zwykle najkrótszy średni czas oczekiwania. Z tej przyczyny CPU ma zwykle najniższy priorytet na magistrali SBI. Dlaczego urządzenie o priorytecie 16 ma zwykle najkrótszy średni czas oczekiwania? W jakich warunkach może to nie być prawdą?

3.10. Nakreśl i wyjaśnij przebiegi czasowe dla operacji zapisu PCI (podobne do rys. 3.23).

Dodatek 3A. Wykresy przebiegów czasowych

W tym rozdziale wykorzystywaliśmy wykresy przebiegów czasowych w celu zilustrowania sekwencji zdarzeń i zależności między zdarzeniami. Czytelnikowi, któremu te wykresy nie są znane, dodatek ten dostarczy niezbędnych wyjaśnień.



Rysunek 3.27. Przebiegi czasowe

Komunikacja między urządzeniami podłączonymi do magistrali zachodzi za pomocą zbioru linii służących do przenoszenia sygnałów. Transmitowane sygnały mogą mieć dwa różne poziomy napięcia, reprezentujące binarne 0 i binarną 1. Wykres czasowy pokazuje poziom sygnału na linii jako funkcję czasu (rys. 3.27a). Zgodnie z przyjętą konwencją poziom sygnału binarnej 1 jest przedstawiany jako wyższy niż binarnego 0. Zwykle binarne 0 ma wartość zaniedbywalną. Jeśli więc żadne dane ani żadne sygnały nie są transmitowane, poziom na linii reprezentuje binarne 0. Przejście sygnału od 0 do 1 jest zwykle określane jako zbocze narastające; przejście od 1 do 0 określa się jako zbocze opadające. Dla uproszczenia, zmiany sygnałów są często przedstawiane jako natychmiastowe. W rzeczywistości zajmują czas niezerowy, jednak jest on zwykle krótki w porównaniu z czasem trwania sygnałów. Na wykresie przebiegu czasowego może się zdarzyć, że pomiędzy interesującymi zdarzeniami upływa zmienna lub przynajmniej nieistotna ilość czasu. Prezentujemy to, stosując przerwę w wykresie.

Sygnały są czasem rysowane w grupach (rys. 3.27b). Na przykład, jeśli dane są przesyłane bajtami, wymaganych jest 8 linii. Ogólnie nie musimy znać dokładnych wartości przesyłanych w takich grupach; musimy tylko wiedzieć, czy sygnały są

obecne, czy nie. Zmiana sygnału na linii może spowodować, że dołączone urządzenie zmieni sygnały na innych liniach. Jeśli na przykład moduł pamięci wykryje sygnał sterujący odczytu (przejście do 0 lub 1), to umieści sygnały danych na liniach danych. Takie związki przyczynowo-skutkowe powodują sekwencje zdarzeń. W celu pokazania tych zależności na wykresach przebiegów czasowych wykorzystuje się strzałki (rys. 3.27c).

Na rysunku 3.27c kreska nad nazwą sygnału wskazuje, że sygnał ten jest aktywny w stanie niskim. Na przykład $\overline{\mathit{Rozkaz}}$ jest czynny (potwierdzony) na poziomie 0V. Oznacza to, że $\overline{\mathit{Rozkaz}} = 0$ jest interpretowany jako logiczna 1 lub „prawda".

Częścią magistrali systemowej jest zwykle linia zegarowa. Do linii zegarowej jest dołączony zegar elektroniczny, wywołujący powtarzalne, regularne sekwencje zmian (rys. 3.27d). Za pomocą sygnału zegarowego mogą być synchronizowane inne zdarzenia.

Rozdział 4

Pamięć podręczna

4.1. Przegląd systemów pamięci komputerowych
Własności systemów pamięci
Hierarchia pamięci
4.2. Zasady działania pamięci podręcznej
4.3. Elementy projektowania pamięci podręcznych
Rozmiar pamięci podręcznej
Funkcja odwzorowywania
Algorytmy zastępowania
Algorytm zapisu
Rozmiar wiersza
Liczba pamięci podręcznych
4.4. Organizacja pamięci podręcznej w Pentium 4 i PowerPC
Organizacja pamięci podręcznej w Pentium 4
Organizacja pamięci podręcznej w PowerPC
4.5. Polecana literatura
4.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania
Dodatek 4A. Charakterystyka wydajności pamięci dwupoziomowych
Lokalność
Działanie pamięci dwupoziomowej
Wydajność

PODSTAWOWE SPOSTRZEŻENIA

- Pamięć komputera jest zorganizowana hierarchicznie. Na poziomie najwyższym (najbliższym procesorowi) znajdują się rejestry procesora. Następnie występuje jeden lub wiele poziomów pamięci podręcznej. Jeśli tych poziomów jest wiele, są one oznaczane jako L1, L2 itd. Na kolejnym poziomie znajduje się pamięć główna, mająca zwykle postać dynamicznej pamięci o swobodnym dostępie (DRAM). Dalszy ciąg hierarchii obejmuje pamięć zewnętrzną, przy czym następnym poziomem jest zwykle dysk twardy, kolejne zaś niższe poziomy składają się z nośników wymiennych, takich jak kasety ZIP, dyski optyczne i taśmy magnetyczne.
- W miarę przesuwania się w dół w hierarchii pamięci maleje koszt przypadający na bit, rośnie pojemność i wydłuża się czas dostępu. Wygodnie byłoby używać jedynie najszybszej pamięci, ponieważ jednak jest ona najdroższa, decydujemy się na kompromis między czasem dostępu a kosztem, wprowadzając więcej powolnych pamięci. Istota sprawy polega na takim zorganizowaniu danych i programów w pamięci, aby potrzebne słowa znajdowały się zwykle w szybszej pamięci.
- Ogólnie rzecz biorąc, jest najbardziej prawdopodobne, że większość przyszłych operacji dostępu do pamięci będzie dotyczyła tych miejsc, do których dostęp miał miejsce ostatnio. Tak więc pamięć podręczna automatycznie zachowuje kopie niektórych spośród używanych ostatnio słów pochodzących z pamięci DRAM. Jeśli pamięć podręczna została właściwie zaprojektowana, procesor będzie potrzebował najczęściej tych słów z pamięci, które znajdują się już w pamięci podręcznej.

Pamięć komputera, pozornie prostą koncepcyjnie, cechuje najszerszy zakres rodzajów, technologii, organizacji, wydajności i kosztów w stosunku do pozostałych składników. Nie istnieje jedna technologia, która optymalnie spełniałaby wymagania systemu komputerowego dotyczące pamięci. W rezultacie typowy system komputerowy zawiera hierarchię podsystemów pamięci: niektóre z nich są wewnętrzne w stosunku do systemu (bezpośrednio dostępne dla procesora), a inne zewnętrzne (dostępne dla procesora poprzez moduł wejścia-wyjścia).

W tym rozdziale i w następnym skoncentrujemy się na składnikach pamięci wewnętrznej, natomiast rozdz. 6 poświęcimy pamięci zewnętrznej. W pierwszym podrozdziale zajmiemy się głównymi własnościami pamięci komputerowych. W pozostałej części rozdziału podamy analizę zasadniczego składnika nowoczesnego składnika wszystkich nowoczesnych systemów komputerowych: pamięci podręcznej.

4.1. Przegląd systemów pamięci komputerowych

Własności systemów pamięci

Złożone zagadnienia pamięci komputerowych można opanować łatwiej, jeśli podzieli się systemy pamięciowe według ich podstawowych własności. Najważniejsze z tych własności są wymienione w tabeli 4.1.

Termin **położenie** (lokacja) użyty w tabeli 4.1 odnosi się do tego, czy pamięć jest wewnętrzna, czy zewnętrzna. Pamięć wewnętrzna jest często identyfikowana z pamięcią główną. Istnieją jednak inne formy pamięci wewnętrznej. Procesor wymaga własnej pamięci lokalnej w postaci rejestrów (patrz np. rys. 2.3). Ponadto, jak zobaczymy, stanowiąca część procesora jednostka sterująca może również potrzebować własnej pamięci wewnętrznej. Dyskusję na temat tych dwóch ostatnich typów pamięci wewnętrznej pozostawiamy do następnych rozdziałów. Pamięć zewnętrzna składa się z peryferyjnych urządzeń pamięciowych, takich jak pamięci dyskowe i taśmowe, które są dostępne dla procesora poprzez sterowniki wejścia-wyjścia.

Tabela 4.1. Podstawowe własności komputerowych systemów pamięciowych

Położenie procesor wewnętrzna (główna) zewnętrzna (pomocnicza)	**Wydajność** czas dostępu czas cyklu szybkość transferu
Pojemność rozmiar słowa liczba słów	**Rodzaj fizyczny** półprzewodnikowa magnetyczna optyczna magnetooptyczna
Jednostka transferu słowo blok	
Sposób dostępu sekwencyjny bezpośredni swobodny skojarzeniowy	**Własności fizyczne** ulotna/nieulotna wymazywalna/niewymazywalna **Organizacja**

Oczywistą własnością pamięci jest jej **pojemność**. W przypadku pamięci wewnętrznej jest ona zwykle wyrażana w bajtach (1 bajt = 8 bitów) lub w słowach. Powszechnymi długościami słów są: 8, 16 i 32 bity. Pojemność pamięci zewnętrznej jest zwykle wyrażana w bajtach.

Parametrem związanym z pojemnością jest **jednostka transferu** (ang. *transfer unit*). W przypadku pamięci wewnętrznej jednostka transferu jest równa liczbie linii danych doprowadzonych do modułu pamięci i wychodzących z niego. Jest ona często równa długości słowa, ale nie musi tak być. W celu wyjaśnienia tego problemu rozważmy trzy powiązane ze sobą pojęcia dotyczące pamięci wewnętrznej:

- **Słowo.** „Naturalna" jednostka organizacji pamięci. Zwykle rozmiar słowa jest równy liczbie bitów wykorzystywanych do reprezentowania liczby lub długości rozkazu. Niestety, jest wiele wyjątków. Na przykład w komputerze CRAY C90 używane są słowa 64-bitowe, jednak liczba całkowita jest reprezentowana za pomocą 46 bitów. VAX ma zdumiewająco różne długości rozkazów, wyrażone jako wielokrotności bajtów, a rozmiar słowa wynosi 32 bity.
- **Jednostka adresowalna.** W wielu systemach jednostką adresowalną jest słowo. Jednak niektóre systemy umożliwiają adresowanie na poziomie bajtów. W każdym przypadku zależność między długością adresu A a liczbą adresowalnych jednostek N jest następująca: $2^A = N$.
- **Jednostka transferu.** W przypadku pamięci głównej jest to liczba bitów jednocześnie odczytywanych z pamięci lub do niej zapisywanych. Jednostka transferu nie musi być równa słowu lub jednostce adresowalnej. W przypadku pamięci zewnętrznej dane są często przekazywane w jednostkach o wiele większych niż słowo, określanych jako bloki.

Jedna z najbardziej widocznych różnic między różnymi rodzajami pamięci dotyczy sposobu dostępu do jednostek danych. Można wyróżnić cztery rodzaje dostępu:

- **Dostęp sekwencyjny.** Pamięć jest zorganizowana za pomocą jednostek danych zwanych rekordami. Dostęp jest możliwy w określonej sekwencji liniowej. Do oddzielania rekordów i do pomocy przy odczycie są wykorzystywane przechowywane informacje adresowe. Odczyt i zapis są wykonywane za pomocą tego samego mechanizmu, przy czym proces ten musi się przenosić z pozycji bieżącej do pozycji pożądanej, przepuszczając i odrzucając każdy rekord pośredni. W rezultacie czas dostępu do różnych rekordów może się bardzo różnić. Pamięci taśmowe, omawiane w rozdz. 6, charakteryzują się właśnie dostępem sekwencyjnym.
- **Dostęp bezpośredni.** Podobnie jak w przypadku dostępu sekwencyjnego, proces odczytu i zapisu w pamięciach o dostępie bezpośrednim jest realizowany za pomocą tego samego mechanizmu. Jednak poszczególne bloki lub rekordy mają unikatowy adres oparty na lokacji fizycznej. Dostęp jest realizowany przez bezpośredni dostęp do najbliższego otoczenia, po którym następuje sekwencyjne poszukiwanie, liczenie lub oczekiwanie w celu osiągnięcia lokacji finalnej. Jak poprzednio czas dostępu jest zmienny. Pamięci dyskowe, które omówimy w rozdz. 6, charakteryzują się dostępem bezpośrednim.
- **Dostęp swobodny.** Każda adresowalna lokacja w pamięci ma unikatowy, fizycznie wbudowany mechanizm adresowania. Czas dostępu do danej lokacji nie zależy od sekwencji poprzednich operacji dostępu i jest stały. Dzięki temu dowolna lokacja może być wybierana swobodnie i jest adresowana i dostępna bezpośrednio. Systemy pamięci głównej wyróżniają się dostępem swobodnym.
- **Dostęp skojarzeniowy.** Jest to rodzaj dostępu swobodnego, który umożliwia porównywanie i specyficzne badanie zgodności wybranych bitów wewnątrz słowa, przy czym jest to czynione dla wszystkich słów jednocześnie. Tak więc słowo jest wyprowadzane raczej na podstawie części swojej zawartości niż na podstawie adresu. Podobnie jak w przypadku zwykłych pamięci o dostępie swobodnym, każda

lokacja ma własny mechanizm adresowania, a czas dostępu jest stały i niezależny od poprzednich operacji dostępu. W pamięciach podręcznych może być używany właśnie dostęp skojarzeniowy.

Z punktu widzenia użytkownika dwiema najważniejszymi własnościami pamięci są pojemność i **wydajność**. Używane są trzy parametry będące miarą wydajności:

- **Czas dostępu.** W przypadku pamięci o dostępie swobodnym jest to czas niezbędny do zrealizowania operacji odczytu lub zapisu, to znaczy czas od chwili doprowadzenia adresu do chwili zmagazynowania lub udostępnienia danych. W przypadku pamięci o dostępie nieswobodnym czas dostępu jest czasem potrzebnym na umieszczenie mechanizmu odczytu-zapisu w pożądanym miejscu.
- **Czas cyklu pamięci.** Pojęcie to było pierwotnie stosowane do pamięci o dostępie swobodnym. Czas cyklu składa się z czasu dostępu oraz z dodatkowego czasu, który musi upłynąć, zanim będzie mógł nastąpić kolejny dostęp. Ten dodatkowy czas może być potrzebny dla zaniku sygnałów przejściowych lub do regeneracji danych, jeśli odczyt jest niszczący.
- **Szybkość przesyłania** (transferu). Jest to szybkość, z jaką dane mogą być wprowadzane do jednostki pamięci lub z niej wyprowadzane. W przypadku pamięci o dostępie swobodnym jest ona równa 1/(czas cyklu).

W przypadku pamięci o dostępie nieswobodnym zachodzi następująca zależność:

$$T_N = T_A + \frac{N}{R}$$

gdzie:
T_N – średni czas odczytu lub zapisu N bitów,
T_A – średni czas dostępu,
N – liczba bitów,
R – szybkość transferu w bitach na sekundę [bit/s].

Wykorzystywane są różne **fizyczne rodzaje** pamięci. Najbardziej powszechne spośród nich obecnie to pamięć półprzewodnikowa, powierzchniowa pamięć magnetyczna w postaci dysków i taśm oraz pamięć optyczna i magneto-optyczna.

Ważne są niektóre **własności fizyczne** urządzeń do przechowywania danych. W pamięci ulotnej informacja zanika naturalnie lub jest tracona po wyłączeniu zasilania. W pamięci nieulotnej informacja raz zapisana nie zmienia się aż do chwili zamierzonej zmiany; zasilanie elektryczne nie jest niezbędne do zachowania informacji. Powierzchniowe pamięci magnetyczne są nieulotne. Pamięć półprzewodnikowa może być albo ulotna, albo nieulotna. Zawartość pamięci niewymazywalnej nie może być zmieniana, z wyjątkiem wypadku zniszczenia jednostki pamięciowej. Pamięć półprzewodnikowa tego typu jest znana jako pamięć stała (*read-only memory* – ROM). Z konieczności pamięć niewymazywalna musi być także nieulotna.

W przypadku pamięci o dostępie swobodnym podstawowym problemem przy projektowaniu jest **organizacja**. Przez organizację rozumiemy tutaj fizyczne uporządkowanie bitów w celu uformowania słów. Jak wyjaśnimy, nie zawsze wykorzystywane jest uporządkowanie, które wydaje się oczywiste.

Hierarchia pamięci

Ograniczenia przy projektowaniu pamięci komputera mogą być podsumowane za pomocą trzech pytań: Ile? Jak szybko? Za ile?

Pytanie „Ile?" jest wciąż w pewnym stopniu otwarte. Jeśli dysponujemy określoną pojemnością, to z pewnością pojawi się zastosowanie, w którym pojemność ta zostanie wykorzystana. Odpowiedź na pytanie „Jak szybko?" jest w pewnym sensie łatwiejsza. Aby osiągnąć największą wydajność, pamięć musi być w stanie nadążyć za procesorem. Znaczy to, że nie chcielibyśmy, aby procesor, wykonując rozkazy, czekał na rozkazy lub argumenty. Ostatnie z pytań również musi być rozważone. W przypadku praktycznego systemu koszt pamięci musi pozostawać w rozsądnej relacji do kosztu pozostałych składników.

Jak można było się spodziewać, istnieją wzajemne zależności między podstawowymi parametrami pamięci, to znaczy między kosztem, pojemnością i czasem dostępu. Jak zawsze przy konstruowaniu systemów pamięciowych są wykorzystywane różne technologie. Jeśli rozpatrujemy dostępne technologie, to zauważamy następujące zależności:

- mniejszy czas dostępu – większy koszt na bit,
- większa pojemność – mniejszy koszt na bit,
- większa pojemność – większy czas dostępu.

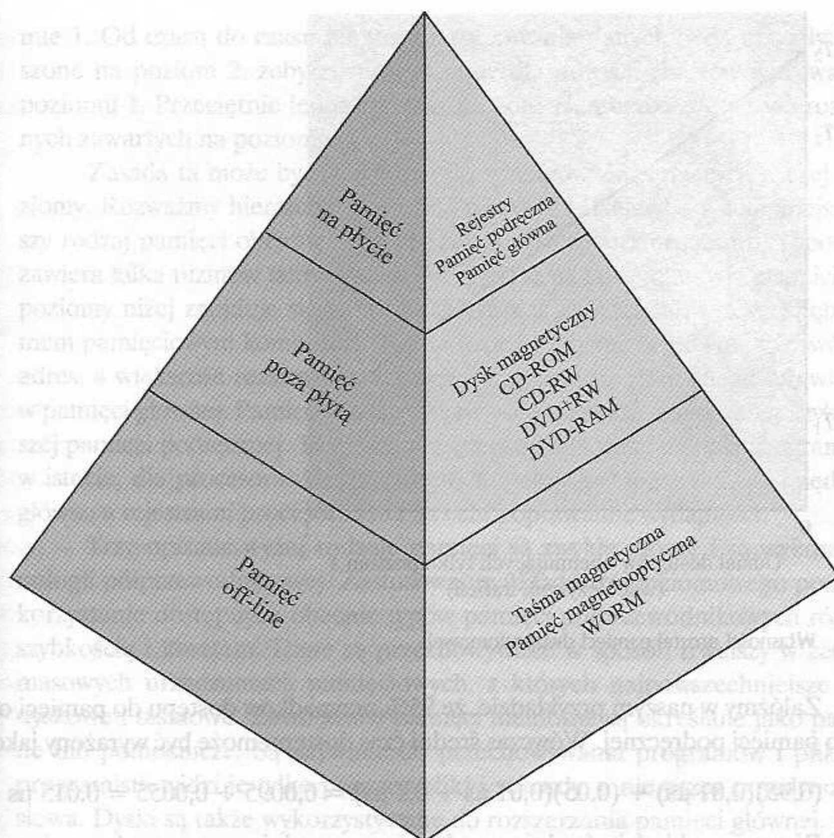
Dylemat, przed którym stoi projektant, jest jasny. Chciałby on użyć technologii pamięci, które umożliwiają wytworzenie pamięci o dużej pojemności, ponieważ po pierwsze są one potrzebne, a po drugie z powodu mniejszego kosztu na bit. Aby jednak osiągnąć wymaganą wydajność, projektant wykorzystuje drogie i o stosunkowo małej pojemności pamięci o krótkim czasie dostępu.

Osiągnięcie kompromisu nie polega na wyborze pojedynczego zespołu pamięci czy też określonej technologii, lecz na wykorzystaniu hierarchii pamięci. Typowa hierarchia jest przedstawiona na rys. 4.1. Rozpatrując tę hierarchię w kierunku od góry do dołu, obserwujemy następujące zjawiska:

(a) malejący koszt na bit,
(b) rosnącą pojemność,
(c) rosnący czas dostępu,
(d) malejącą częstotliwość dostępu do pamięci przez procesor.

Tak więc mniejsze, droższe i szybsze pamięci są uzupełniane przez pamięci większe, tańsze i powolniejsze. Kluczem do sukcesu przy projektowaniu organizacji jest ostatnie z wymienionych zjawisk, a mianowicie zmniejszanie częstotliwości dostępu. Przedyskutujemy tę koncepcję szczegółowo podczas omawiania pamięci podręcznej w dalszej części tego rozdziału oraz pamięci wirtualnej w rozdz. 8, tutaj jednak przedstawimy krótkie wyjaśnienie.

Załóżmy, że procesor ma dostęp do dwóch poziomów pamięci. Poziom pierwszy zawiera 1000 słów i ma czas dostępu 0,01 μs. Poziom 2 zawiera 100 000 słów i ma czas dostępu 0,1 μs. Załóżmy dalej, że jeśli poszukiwane słowo znajduje się na

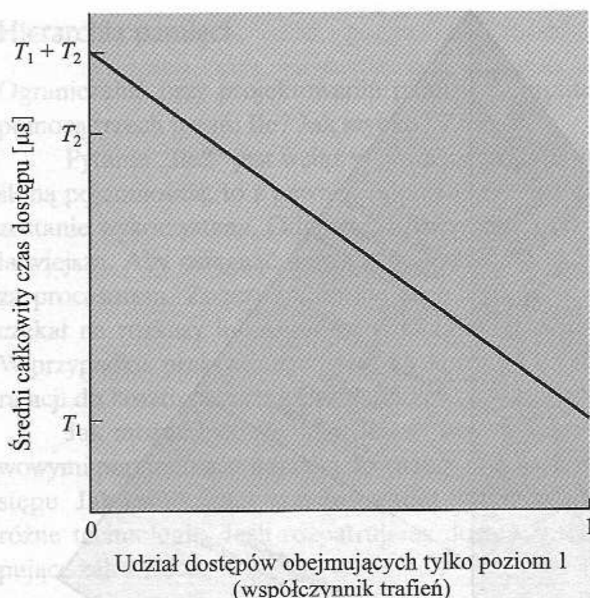


Rysunek 4.1. Hierarchia pamięci

poziomie 1, to procesor ma do niego dostęp bezpośredni. Jeśli natomiast jest ono na poziomie 2, to słowo jest najpierw przenoszone na poziom 1, a następnie udostępniane procesorowi. Dla uproszczenia zaniedbajmy czas wymagany do tego, aby procesor określił, czy słowo znajduje się na pierwszym, czy też na drugim poziomie. Na rysunku 4.2 jest widoczny ogólny kształt krzywej odnoszącej się do tej sytuacji. Średni całkowity czas dostępu jako funkcja procentu czasu, w ciągu którego słowo jest znajdywane od razu na poziomie 1. Pokazano na nim średni czas dostępu do pamięci dwupoziomowej jako funkcję współczynnika trafienia H, gdzie:

H – część wszystkich dostępów do pamięci, która jest realizowana w pamięci szybszej (np. podręcznej),
T_1 – czas dostępu do poziomu 1,
T_2 – czas dostępu do poziomu 2.

Jak można zauważyć, przy wysokim udziale bezpośrednich dostępów do poziomu 1, średni całkowity czas dostępu jest o wiele bliższy czasowi charakterystycznemu dla poziomu 1 niż dla 2.



Rysunek 4.2. Własności prostej pamięci dwupoziomowej

Załóżmy w naszym przykładzie, że 95% przypadków dostępu do pamięci odnosi się do pamięci podręcznej. Wówczas średni czas dostępu może być wyrażony jako

$$(0,95)(0,01\ \mu s) + (0,05)(0,01\ \mu s + 0,1\ \mu s) = 0,0095 + 0,0055 = 0,015\ \mu s$$

W tym przykładzie średni czas dostępu – zgodnie z naszym życzeniem – jest znacznie bliższy 0,01 μs, niż 0,1 μs. Zastosowanie pamięci dwupoziomowej w celu zmniejszenia średniego czasu dostępu w zasadzie się sprawdza, jednak tylko wówczas, jeśli będą spełnione warunki od (a) do (d). Używając różnych technologii, uzyskujemy spektrum systemów pamięci, które spełnia warunki od (a) do (c). Na szczęście warunek (d) jest również ogólnie słuszny.

Podstawą słuszności warunku (d) jest zasada **lokalności odniesień** (*locality of reference*) [DENN68]. Podczas wykonywania programu, odniesienia do pamięci (zarówno dotyczące danych, jak i rozkazów) mają tendencję do gromadzenia się. Programy typowo zawierają pewną liczbę pętli iteracyjnych i podprogramów. Gdy jest uruchomiona pętla lub podprogram, występują powtarzające się odniesienia do niewielkiego zestawu rozkazów. Podobnie, operacje na tablicach lub ciągach danych polegają na dostępie do zgrupowanego zespołu słów. W ciągu długiego czasu wykorzystywane grupy zmieniają się, jednak w krótkim czasie procesor pracuje z ustalonymi grupami odniesień do pamięci.

W tej sytuacji jest możliwa taka organizacja układu danych w hierarchii pamięci, żeby udział dostępów do każdego kolejnego niższego poziomu był istotnie mniejszy niż do poziomu wyższego. Rozpatrzmy dwupoziomowy przykład przedstawiony powyżej. Przyjmijmy, że poziom 2 pamięci zawiera wszystkie rozkazy i dane programu. Bieżące ugrupowania danych mogą być czasowo umieszczane na pozio-

mie 1. Od czasu do czasu niektóre ugrupowania danych będą musiały być przenoszone na poziom 2, żeby zwolnić miejsce dla nowych zbiorów wprowadzanych do poziomu 1. Przeciętnie jednak większość odniesień będzie dotyczyła rozkazów i danych zawartych na poziomie 1.

Zasada ta może być stosowana do systemów zawierających więcej niż dwa poziomy. Rozważmy hierarchię pokazaną na rys. 4.1. Najszybszy, najmniejszy i najdroższy rodzaj pamięci obejmuje rejestry zawarte wewnątrz procesora. Typowo procesor zawiera kilka tuzinów takich rejestrów, chociaż niektóre maszyny mają ich setki. Dwa poziomy niżej znajduje się pamięć główna, będąca zasadniczym wewnętrznym systemem pamięciowym komputera. Każda lokacja w pamięci głównej ma swój unikatowy adres, a większość rozkazów maszynowych odnosi się do jednego lub wielu adresów w pamięci głównej. Pamięć główna jest zwykle poszerzana za pomocą szybszej i mniejszej pamięci podręcznej. Ta ostatnia nie jest zwykle widzialna dla programisty czy też, w istocie, dla procesora. Urządzenie to stopniuje przepływ danych między pamięcią główną a rejestrami procesora i ma na celu poprawienie wydajności.

Trzy opisane wyżej rodzaje pamięci są zwykle ulotne i są wykonane w technologii półprzewodnikowej. Zastosowanie układu trójpoziomowego pozwala na wykorzystanie dostępnych obecnie typów pamięci półprzewodnikowych różniących się szybkością i kosztem. Dane są przechowywane w sposób trwalszy w zewnętrznych, masowych urządzeniach pamięciowych, z których najpowszechniejsze są pamięci dyskowe i taśmowe. Zewnętrzne pamięci nieulotne są określane jako pamięci wtórne lub pomocnicze. Są używane do przechowywania programów i plików danych; programista widzi je tylko poprzez pliki i rekordy, a nie przez pojedyncze bajty czy słowa. Dyski są także wykorzystywane do rozszerzania pamięci głównej, jako pamięci wirtualne, którymi zajmiemy się w rozdz. 8.

W hierarchii mogą występować również inne rodzaje pamięci. Na przykład duże komputery IBM zawierają pewną postać pamięci wewnętrznej znaną jako *pamięć rozszerzona* (*expanded storage*). Wykorzystuje się w niej wolniejszą i tańszą technologię półprzewodnikową niż w pamięci głównej. Ściśle mówiąc, pamięć ta nie tworzy hierarchii, lecz stanowi rodzaj bocznej gałęzi: dane mogą być przenoszone między pamięcią główną a pamięcią rozszerzoną, jednak nie między pamięcią rozszerzoną a pamięcią zewnętrzną. Do innych postaci pamięci wtórnych należą dyski optyczne i magnetooptyczne. Wreszcie, dodatkowe poziomy mogą być skutecznie dodane do hierarchii za pomocą oprogramowania. Część pamięci głównej może być wykorzystywana jako bufor do tymczasowego przechowywania danych, które następnie mają być odczytane i wyprowadzone na dysk. Taka technika, określana czasem jako dyskowa pamięć podręczna[1], poprawia wydajność na dwa sposoby:

- Zapisy dyskowe są grupowane. W miejsce wielu transferów niewielkich ilości danych mamy do czynienia z kilkoma wielkimi transferami. Poprawia to wydajność dysku i minimalizuje zaangażowanie procesora.

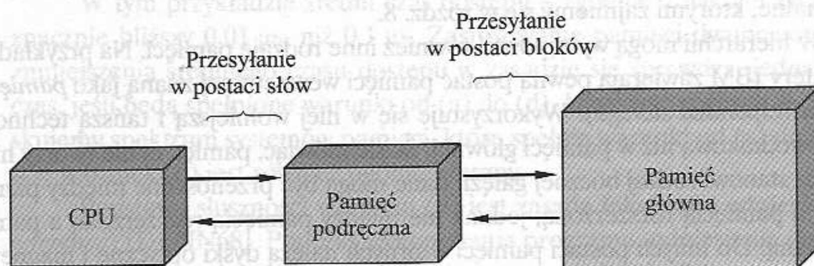
[1] Dyskowa pamięć podręczna jest na ogół rozwiązaniem czysto programowym i nie jest analizowana w tej książce. Omówienie tej pamięci można znaleźć w [STAL01].

- Pewne dane przeznaczone do odczytania mogą być określone przez program jeszcze przed następnym sięgnięciem do dysku. W tym przypadku dane te mogą być wydobyte szybko z programowej pamięci podręcznej, a nie powoli z dysku.

W dodatku 4A jest przedyskutowany wpływ wielopoziomowych struktur pamięciowych na wydajność.

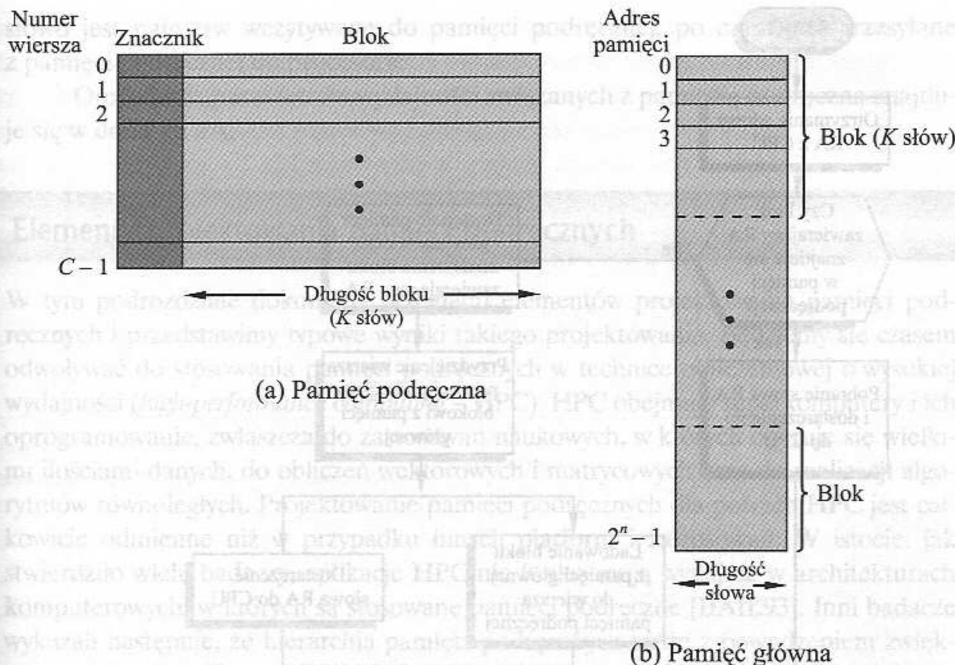
4.2. Zasady działania pamięci podręcznej

Pamięci podręczne stosuje się w celu uzyskania pamięci o takiej szybkości, jaką mają najszybsze osiągalne pamięci, z jednoczesnym uzyskaniem dużego rozmiaru pamięci w cenie tańszych rodzajów pamięci półprzewodnikowych. Koncepcja ta jest zilustrowana na rys. 4.3. Występuje tu względnie duża i wolniejsza pamięć główna obok mniejszej i szybszej pamięci podręcznej. Pamięć podręczna zawiera kopię części zawartości pamięci głównej. Gdy procesor zamierza odczytać słowo w pamięci, najpierw następuje sprawdzenie, czy słowo to nie znajduje się w pamięci podręcznej. Jeśli tak, to słowo jest dostarczane do procesora. Jeśli nie, to blok pamięci głównej zawierający ustaloną liczbę słów jest wczytywany do pamięci podręcznej, a następnie słowo jest dostarczane do procesora. Ze względu na zjawisko lokalności odniesień, jeśli blok danych został pobrany do pamięci podręcznej w celu zaspokojenia pojedynczego odniesienia do pamięci, jest prawdopodobne, że przyszłe odniesienia będą dotyczyły innych słów zawartych w tym samym bloku.



Rysunek 4.3. Pamięć podręczna i pamięć główna

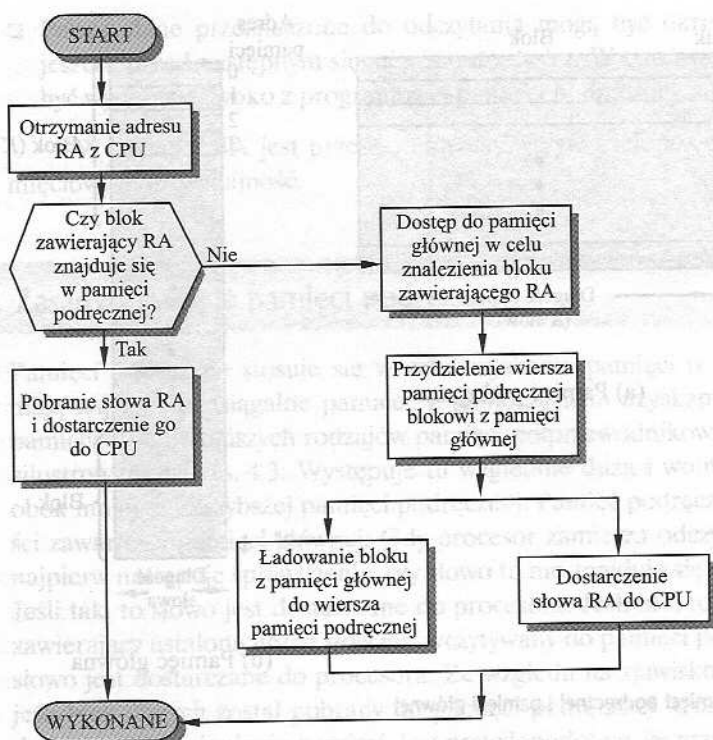
Na rysunku 4.4 jest pokazana struktura systemu pamięć podręczna–pamięć główna. Pamięć główna składa się z 2^n adresowalnych słów, przy czym każde słowo ma jednoznaczny adres n-bitowy. Aby było możliwe odwzorowywanie, pamięć ta składa się z pewnej liczby bloków o stałej długości, zawierających K słów każdy. Tak więc występuje $M = 2^n/K$ bloków. Pamięć podręczna zawiera C **wierszy** zawierających K słów każdy, a liczba wierszy jest znacząco mniejsza od liczby bloków w pamięci głównej ($C \ll M$). W każdej chwili pewien zespół bloków pamięci pozostaje w wierszach pamięci podręcznej. Jeśli słowo w bloku pamięci jest odczytywane, to odpowiedni blok jest przenoszony do jednego z wierszy pamięci podręcznej. Po-



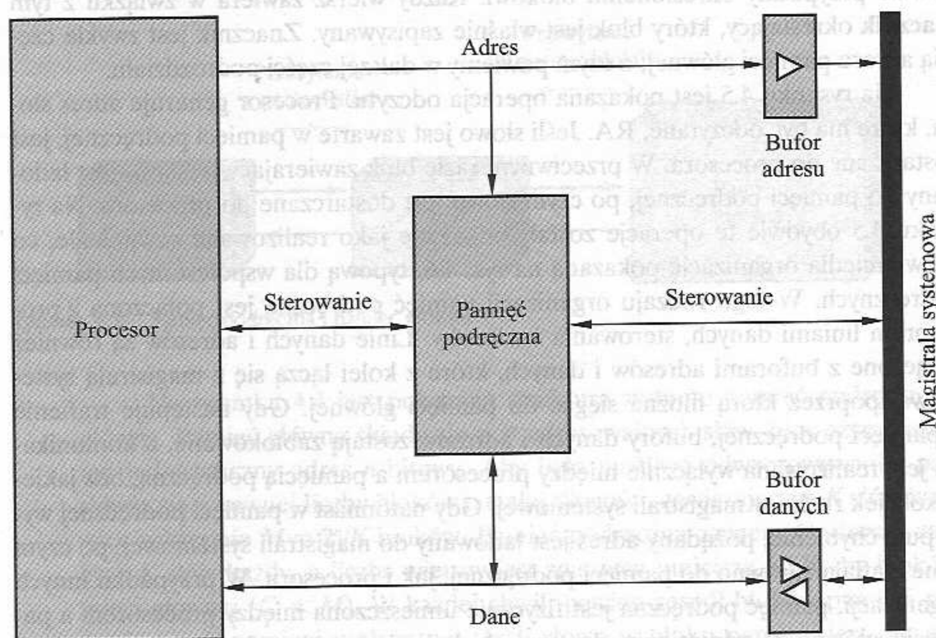
Rysunek 4.4. Struktura pamięci podręcznej i pamięci głównej

nieważ bloków jest więcej niż wierszy, określony wiersz nie może być jednoznacznie i trwale przypisany określonemu blokowi. Każdy wiersz zawiera w związku z tym **znacznik** określający, który blok jest właśnie zapisywany. Znacznik jest zwykle częścią adresu pamięci głównej, o czym powiemy w dalszej części podrozdziału.

Na rysunku 4.5 jest pokazana operacja odczytu. Procesor generuje adres słowa, które ma być odczytane, RA. Jeśli słowo jest zawarte w pamięci podręcznej, jest dostarczane do procesora. W przeciwnym razie blok zawierający to słowo jest ładowany do pamięci podręcznej, po czym słowo jest dostarczane do procesora. Na rysunku 4.5 obydwie te operacje zostały pokazane jako realizowane równolegle, co odzwierciedla organizację pokazaną na rys. 4.6, typową dla współczesnych pamięci podręcznych. W tego rodzaju organizacji pamięć podręczna jest połączona z procesorem liniami danych, sterowania i adresów. Linie danych i adresów są również połączone z buforami adresów i danych, które z kolei łączą się z magistralą systemową, poprzez którą można sięgać do pamięci głównej. Gdy następuje trafienie w pamięci podręcznej, bufory danych i adresów zostają zablokowane, a komunikacja jest realizowana wyłącznie między procesorem a pamięcią podręczną, bez jakiegokolwiek ruchu w magistrali systemowej. Gdy natomiast w pamięci podręcznej występuje chybienie, pożądany adres jest ładowany do magistrali systemowej, po czym dane trafiają zarówno do pamięci podręcznej, jak i procesora. W przypadku innych organizacji, pamięć podręczna jest fizycznie umieszczona między procesorem a pamięcią główną w odniesieniu do wszystkich linii danych, adresów i sterowania. Jeśli w tym ostatnim przypadku następuje chybienie w pamięci podręcznej, pożądane



Rysunek 4.5. Operacja odczytu z pamięci podręcznej



Rysunek 4.6. Typowa organizacja pamięci podręcznej

słowo jest najpierw wczytywane do pamięci podręcznej, po czym jest przesyłane z pamięci podręcznej do procesora.

Omówienie parametrów wydajności związanych z pamięcią podręczną znajduje się w dodatku 4A.

4.3. Elementy projektowania pamięci podręcznych

W tym podrozdziale dokonamy przeglądu elementów projektowania pamięci podręcznych i przedstawimy typowe wyniki takiego projektowania. Będziemy się czasem odwoływać do stosowania pamięci podręcznych w technice obliczeniowej o wysokiej wydajności (*high-performance computing* – HPC). HPC obejmuje superkomputery i ich oprogramowanie, zwłaszcza do zastosowań naukowych, w których operuje się wielkimi ilościami danych, do obliczeń wektorowych i matrycowych oraz do realizacji algorytmów równoległych. Projektowanie pamięci podręcznych dla potrzeb HPC jest całkowicie odmienne niż w przypadku innych platform i zastosowań. W istocie, jak stwierdziło wielu badaczy, aplikacje HPC nie funkcjonują wydajnie w architekturach komputerowych, w których są stosowane pamięci podręczne [BAIL93]. Inni badacze wykazali następnie, że hierarchia pamięci podręcznych może z powodzeniem zwiększyć wydajność, jeśli oprogramowanie aplikacyjne zostanie do takiego rozwiązania dostosowane [WANG99, PRES01][2].

Chociaż istnieje wiele implementacji pamięci podręcznych, niewiele jest podstawowych elementów projektowania, które służą do klasyfikowania i różnicowania architektury pamięci podręcznych. Są one wymienione w tabeli 4.2.

Tabela 4.2. Elementy projektowania pamięci podręcznych

Rozmiar pamięci podręcznej **Odwzorowywanie** Bezpośrednie Skojarzeniowe (asocjacyjne) Sekcyjno-skojarzeniowe	**Metoda zapisu** Zapis jednoczesny Zapis opóźniony Zapis jednorazowy
Algorytm zastępowania Najdawniej używany (LRU) Pierwszy na wejściu – pierwszy na wyjściu (FIFO) Najrzadziej używany (LFU) Przypadkowy	**Rozmiar wiersza** **Liczba pamięci podręcznych** Jedno- lub dwupoziomowa Jednorodna lub podzielona

Rozmiar pamięci podręcznej

Pierwszy element, rozmiar pamięci podręcznej, już omówiliśmy. Chcielibyśmy, żeby rozmiar pamięci podręcznej był na tyle mały, żeby ogólny przeciętny koszt na bit był zbliżony do kosztu samej pamięci głównej, oraz na tyle duży, żeby ogólny przeciętny czas dostępu był zbliżony do czasu dostępu samej pamięci podręcznej. Występuje

[2] Ogólne omówienie HPC – zobacz [DOWD98].

jeszcze parę innych powodów minimalizowania rozmiarów pamięci podręcznej. Im większa jest pamięć podręczna, tym większa jest też liczba bramek służących do adresowania tej pamięci. W rezultacie duże pamięci podręczne są nieco wolniejsze niż małe – nawet jeśli są wykonane za pomocą tej samej technologii układów scalonych i umieszczone w tym samym miejscu w mikroukładzie i w obwodzie drukowanym. Rozmiar pamięci podręcznej jest również ograniczony dostępną powierzchnią mikroukładu i obwodu drukowanego. Ponieważ wydajność pamięci podręcznej jest bardzo wrażliwa na naturę realizowanych zadań, nie jest możliwe ścisłe określenie optymalnego rozmiaru tej pamięci. W tabeli 4.3 zostały zestawione wielkości pamięci podręcznych niektórych współczesnych i dawniej używanych procesorów.

Tabela 4.3. Rozmiary pamięci podręcznych wybranych procesorów

Procesor	Rodzaj	Rok wprowadzenia na rynek	Pamięć podręczna L1[a]	Pamięć podręczna L2	Pamięć podręczna L3
IBM 360/85	Duży komputer	1968	16÷32 KB	–	–
PDP-11/70	Minikomputer	1975	1 KB	–	–
VAX 11/780	Minikomputer	1978	16 KB	–	–
IBM 3033	Duży komputer	1978	64 KB	–	–
IBM 3090	Duży komputer	1985	128÷256 KB	–	–
Intel 80486	PC	1989	8 KB	–	–
Pentium	PC	1993	8 KB/8 KB	256÷512 KB	–
PowerPC 601	PC	1993	32 KB	–	–
PowerPC 620	PC	1996	32 KB/32 KB	–	–
PowerPC G4	PC/serwer	1999	32 KB/32 KB	256 KB÷1 MB	2 MB
IBM S/390 G4	Duży komputer	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Duży komputer	1999	256 KB	8 MB	–
Pentium 4	PC/serwer	2000	8 KB/8 KB	256 KB	–
IBM SP	Serwery o dużej wydajności i superkomputery	2000	64 KB/32 KB	8 MB	–
CRAY MTA[b]	PC/serwer	2001	16 KB/16 KB	96 KB	4 MB
Itanium	PC/serwer	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	Serwery o dużej wydajności	2001	32 kB/32 kB	4 MB	–

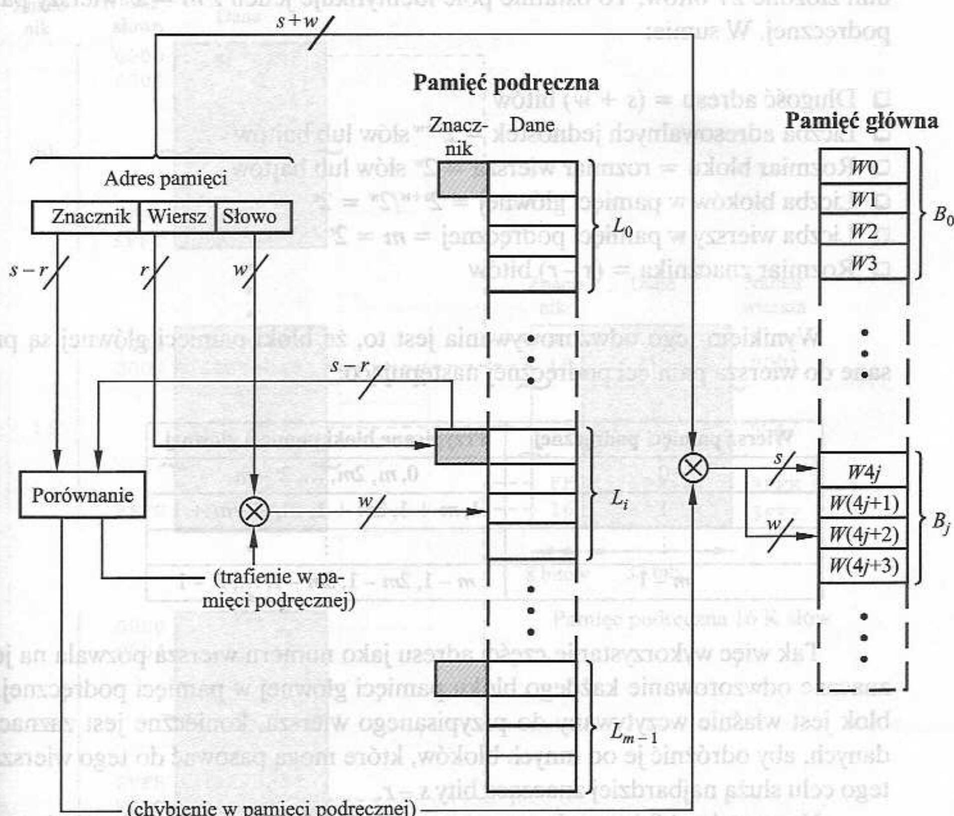
[a] Dwie wartości oddzielone ukośnikiem odnoszą się do pamięci podręcznych rozkazów i danych.
[b] Obydwie pamięci podręczne są przeznaczone tylko dla rozkazów; brak pamięci podręcznej danych.

Funkcja odwzorowywania

Ponieważ wierszy w pamięci podręcznej jest mniej niż bloków pamięci głównej, wymagany jest algorytm odwzorowywania bloków pamięci głównej w wierszach pamięci podręcznej. Ponadto potrzebne są środki do określania, który blok pamięci głównej aktualnie zajmuje wiersz pamięci podręcznej. Wybór funkcji odwzorowywania dyktuje

organizację pamięci podręcznej. Mogą być wykorzystywane trzy metody: bezpośrednia, skojarzeniowa (asocjacyjna) i sekcyjno-skojarzeniowa. Omówimy je kolejno. W każdym przypadku rozpatrzymy strukturę ogólną, a następnie szczególny przykład. Dla wszystkich trzech przypadków przykład zawiera następujące elementy:

- Pamięć podręczna może przechowywać 64 KB.
- Dane są przenoszone między pamięcią główną a pamięcią podręczną w blokach po 4 bajty. Oznacza to, że pamięć podręczna jest zorganizowana w postaci 16 K = 2^{14} wierszy po 4 bajty każdy.
- Pamięć główna składa się z 16 MB, przy czym każdy bajt jest bezpośrednio adresowalny za pomocą 24-bitowego adresu (2^{24} = 16 M). Tak więc, aby umożliwić odwzorowywanie, możemy traktować pamięć główną jako składającą się z 4 M bloków po 4 bajty każdy.



Rysunek 4.7. Organizacja pamięci podręcznej o bezpośrednim odwzorowaniu [HWAN93]

Najprostsza metoda, znana jako **odwzorowanie bezpośrednie**, polega na odwzorowywaniu każdego bloku pamięci głównej na tylko jeden możliwy wiersz pamięci podręcznej. Schemat tej metody jest pokazany na rys. 4.7. Odwzorowywanie jest wyrażane jako

$i = j$ modulo m

gdzie:
i – numer wiersza pamięci podręcznej,
j – numer bloku pamięci głównej,
m – liczba wierszy w pamięci podręcznej.

Funkcja odwzorowywania może być z łatwością realizowana za pomocą adresu. W celu uzyskania dostępu do pamięci podręcznej każdy adres pamięci głównej może być widziany jako składający się z 3 pól. Najmniej znaczące bity w określają jednoznacznie słowo lub bajt w bloku pamięci głównej; w najnowocześniejszych maszynach adres jest formułowany na poziomie bajtów. Pozostałych s bitów określa jeden z 2^s bloków pamięci głównej. Układy logiczne pamięci podręcznej interpretują te s bitów jako znaczniki w postaci $s - r$ bitów (najbardziej znacząca część) oraz pole linii złożone z r bitów. To ostatnie pole identyfikuje jeden z $m = 2^r$ wierszy pamięci podręcznej. W sumie:

- Długość adresu = $(s + w)$ bitów
- Liczba adresowalnych jednostek = 2^{s+w} słów lub bajtów
- Rozmiar bloku = rozmiar wiersza = 2^w słów lub bajtów
- Liczba bloków w pamięci głównej = $2^{s+w}/2^w = 2^s$
- Liczba wierszy w pamięci podręcznej = $m = 2^r$
- Rozmiar znacznika = $(s - r)$ bitów

Wynikiem tego odwzorowywania jest to, że bloki pamięci głównej są przypisane do wiersza pamięci podręcznej następująco:

Wiersz pamięci podręcznej	**Przypisane bloki pamięci głównej**
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

Tak więc wykorzystanie części adresu jako numeru wiersza pozwala na jednoznaczne odwzorowanie każdego bloku pamięci głównej w pamięci podręcznej. Gdy blok jest właśnie wczytywany do przypisanego wiersza, konieczne jest zaznaczenie danych, aby odróżnić je od innych bloków, które mogą pasować do tego wiersza. Do tego celu służą najbardziej znaczące bity $s - r$.

Na rysunku 4.8 jest pokazany nasz przykładowy system stosujący odwzorowanie bezpośrednie[3]. W tym przykładzie $m = 16\text{ K} = 2^{14}$ oraz $i = j$ modulo 2^{14}. Odwzorowanie jest następujące:

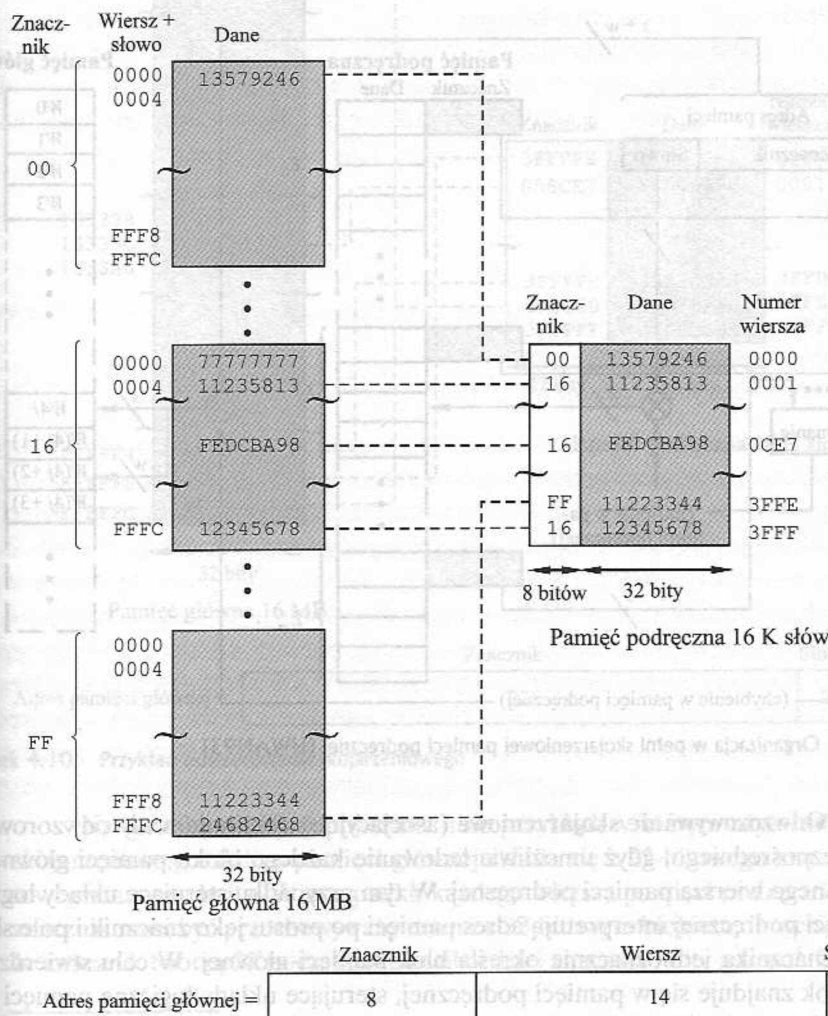
---

[3] Na tym i na następnych rysunkach adresy i wartości pamięci są przedstawiane w notacji szesnastkowej. Podstawowe informacje na temat systemów liczenia (dziesiętnego, binarnego, szesnastkowego) są zawarte w dodatku B.

Wiersz w pamięci podręcznej	Przypisane bloki pamięci głównej
0	000000, 010000, ..., FF0000
1	000004, 010004, ..., FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, ..., FFFFFC

Zauważmy, że żadne dwa bloki odwzorowane w tym samym wierszu nie mają takiego samego znacznika. Bloki 000000, 010000, ..., FF0000 mają znaczniki odpowiednio 00, 01, ..., FF.

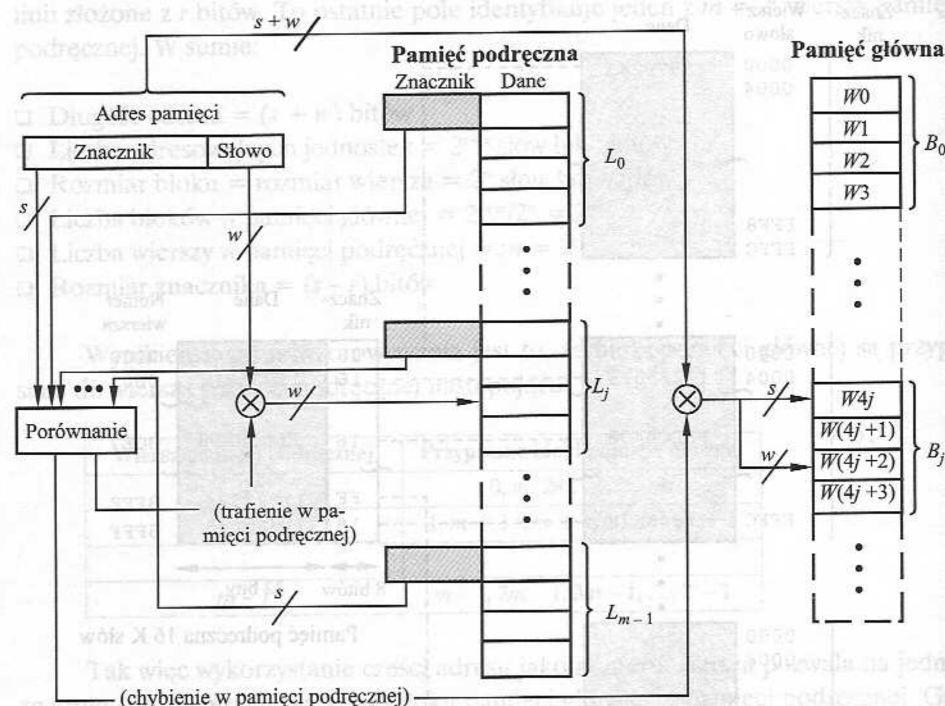
Wracając do rys. 4.5, operację odczytu możemy przedstawić następująco. System pamięci podręcznej jest prezentowany za pomocą adresów 24-bitowych. Czternastobitowy numer wiersza jest wykorzystywany jako indeks dostępu do określonego



Rysunek 4.8. Przykład odwzorowania bezpośredniego

wiersza w pamięci podręcznej. Jeśli 8-bitowy znacznik jest zgodny ze znacznikiem aktualnie przechowywanym w tym wierszu, to 2-bitowy numer słowa służy do wyboru jednego z 4 bajtów w tym wierszu. W przeciwnym razie 22-bitowe pole obejmujące znacznik i wiersz jest wykorzystywane do pobrania bloku z pamięci głównej. Adres użyty do tego pobierania składa się z tych 22 bitów powiązanych z dwoma bitami zerowymi, tak więc na granicy bloku są pobierane 4 bajty.

Metoda odwzorowywania bezpośredniego jest prosta i tania przy wdrażaniu. Jej główną wadą jest to, że dla danego bloku istnieje stała lokalizacja w pamięci podręcznej. W rezultacie, jeśli program będzie się często odwoływał do słów z dwóch różnych bloków przypisanych do tego samego wiersza, to bloki te będą ciągle przenoszone do pamięci podręcznej, co zmniejszy ogólną szybkość (zjawisko zwane *zaśmiecaniem*, ang. *trashing*).

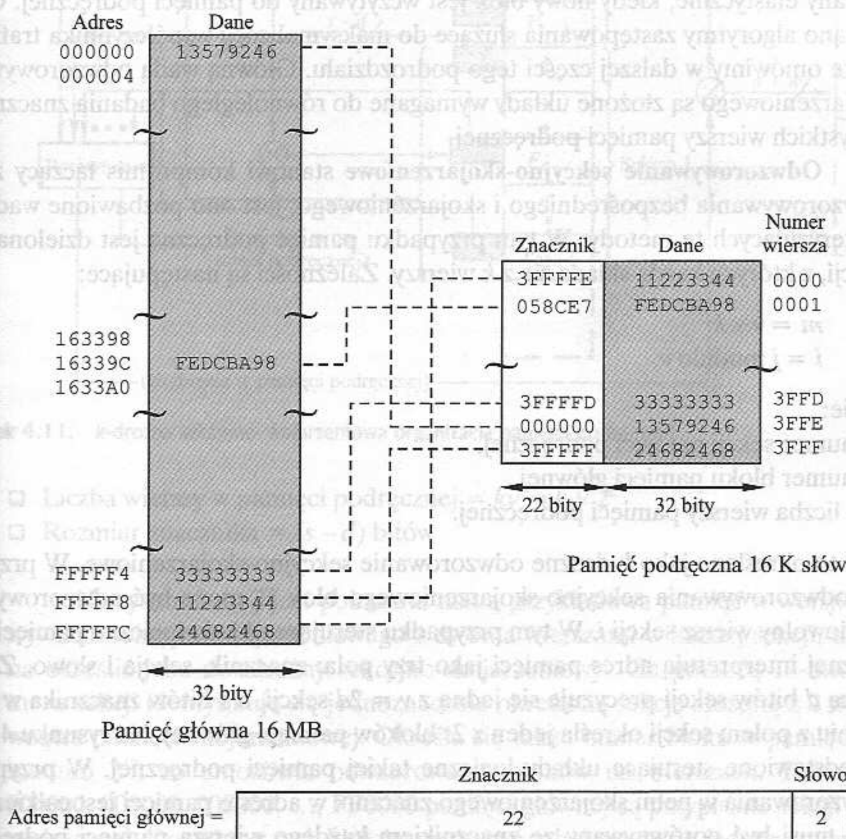


Rysunek 4.9. Organizacja w pełni skojarzeniowej pamięci podręcznej [HWAN93]

Odwzorowywanie skojarzeniowe (asocjacyjne) eliminuje wady odwzorowywania bezpośredniego, gdyż umożliwia ładowanie każdego bloku pamięci głównej do dowolnego wiersza pamięci podręcznej. W tym przypadku sterujące układy logiczne pamięci podręcznej interpretują adres pamięci po prostu jako znacznik i pole słowa. Pole znacznika jednoznacznie określa blok pamięci głównej. W celu stwierdzenia, czy blok znajduje się w pamięci podręcznej, sterujące układy logiczne pamięci podręcznej muszą jednocześnie zbadać zgodność znacznika każdego wiersza. Schemat logiczny został pokazany na rys. 4.9. Zauważmy, że żadne pole adresu nie odpowia-

da numerowi wiersza, więc liczba wierszy w pamięci podręcznej nie jest wyznaczona przez format adresu. W sumie:

- Długość adresu = $(s + w)$ bitów
- Liczba jednostek adresowalnych = 2^{s+w} słów lub bajtów
- Rozmiar bloku = rozmiar wiersza = 2^w słów lub bajtów
- Liczba bloków w pamięci głównej = $2^{s+w}/2^w = 2^s$
- Liczba wierszy w pamięci podręcznej = nieokreślona
- Rozmiar znacznika = s bitów



Rysunek 4.10. Przykład odwzorowania skojarzeniowego

Na rysunku 4.10 jest pokazany nasz przykład, w którym użyto odwzorowania skojarzeniowego. Adres pamięci głównej składa się z 22-bitowego znacznika i 2-bitowego numeru bajta. W przypadku każdego wiersza pamięci podręcznej 22-bitowy znacznik musi być przechowywany razem z 32-bitowym blokiem danych. Zauważmy, że znacznik tworzą 22 początkowe (najbardziej znaczące) bity adresu[4]. Zatem 24-bi-

[4] Na rysunku 4.10 22-bitowy znacznik jest reprezentowany przez 6-cyfrową liczbę szesnastkową. Najbardziej znacząca cyfra szesnastkowa ma więc w rzeczywistości długość zaledwie 2 bitów.

towy adres szesnastkowy 16339C ma znacznik 22-bitowy 058CE7. Można to z łatwością dostrzec w notacji binarnej:

adres pamięci	0001	0110	0011	0011	1001	1100	(binarny)
	1	6	3	3	9	C	(szesnastkowy)
znacznik (22 początkowych bitów)	00	0101	1000	1100	1110	0111	(binarny)
	0	5	8	C	E	7	(szesnastkowy)

W przypadku odwzorowywania skojarzeniowego blok do zastąpienia jest wybierany elastycznie, kiedy nowy blok jest wczytywany do pamięci podręcznej. Opracowano algorytmy zastępowania służące do maksymalizacji współczynnika trafienia, które omówimy w dalszej części tego podrozdziału. Główną wadą odwzorowywania skojarzeniowego są złożone układy wymagane do równoległego badania znaczników wszystkich wierszy pamięci podręcznej.

Odwzorowywanie sekcyjno-skojarzeniowe stanowi kompromis łączący zalety odwzorowywania bezpośredniego i skojarzeniowego; jest ono pozbawione wad charakteryzujących te metody. W tym przypadku pamięć podręczna jest dzielona na v sekcji, z których każda składa się z k wierszy. Zależności są następujące:

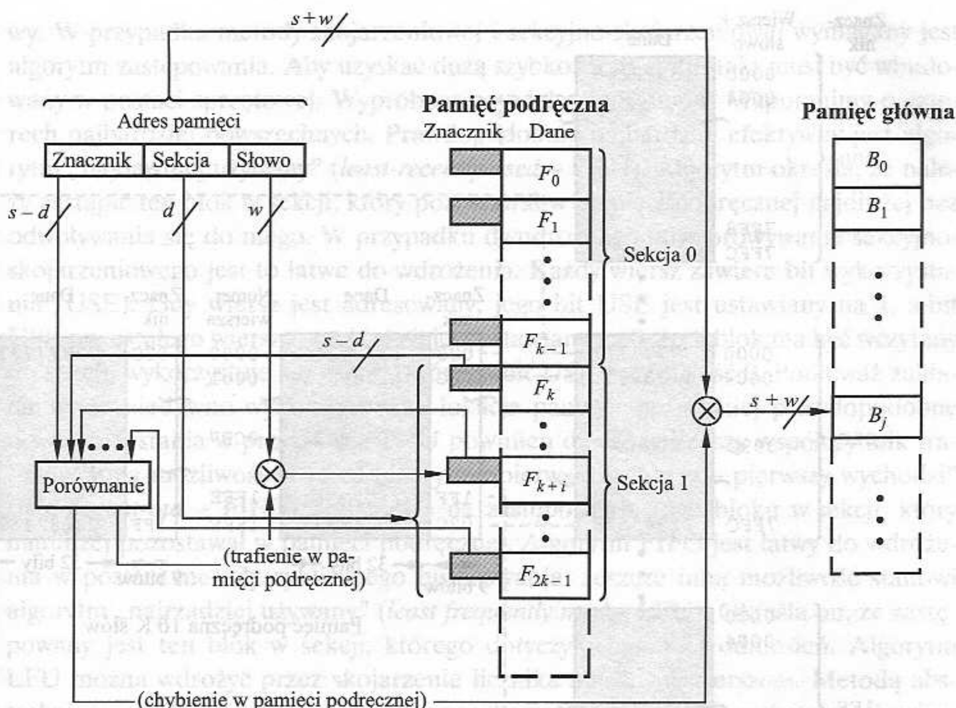
$m = v \times k$

$i = j$ modulo v

gdzie:
i – numer sekcji pamięci podręcznej,
j – numer bloku pamięci głównej,
m – liczba wierszy pamięci podręcznej.

Jest to określane jako k-drożne odwzorowanie sekcyjno-skojarzeniowe. W przypadku odwzorowywania sekcyjno-skojarzeniowego blok B_j może być odwzorowywany na dowolny wiersz sekcji i. W tym przypadku sterujące układy logiczne pamięci podręcznej interpretują adres pamięci jako trzy pola: znacznik, sekcja i słowo. Za pomocą d bitów sekcji precyzuje się jedną z $v = 2d$ sekcji, a s bitów znacznika w połączeniu z polem sekcji określa jeden z $2s$ bloków pamięci głównej. Na rysunku 4.11 są przedstawione sterujące układy logiczne takiej pamięci podręcznej. W przypadku odwzorowania w pełni skojarzeniowego znacznik w adresie pamięci jest całkiem duży i musi być porównywany ze znacznikiem każdego wiersza pamięci podręcznej. Natomiast w przypadku k-drożnego odwzorowania sekcyjno-skojarzeniowego znacznik w adresie pamięci jest znacznie mniejszy i jest porównywany jedynie z k znacznikami w pojedynczej sekcji. W sumie:

- Długość adresu = $(s + w)$ bitów
- Liczba jednostek adresowalnych = 2^{s+w} słów lub bajtów
- Rozmiar bloku = rozmiar wiersza = 2^w słów lub bajtów
- Liczba bloków w pamięci głównej = $2^{s+w}/2^w = 2^s$
- Liczba wierszy w sekcji = k
- Liczba sekcji $v = 2^d$



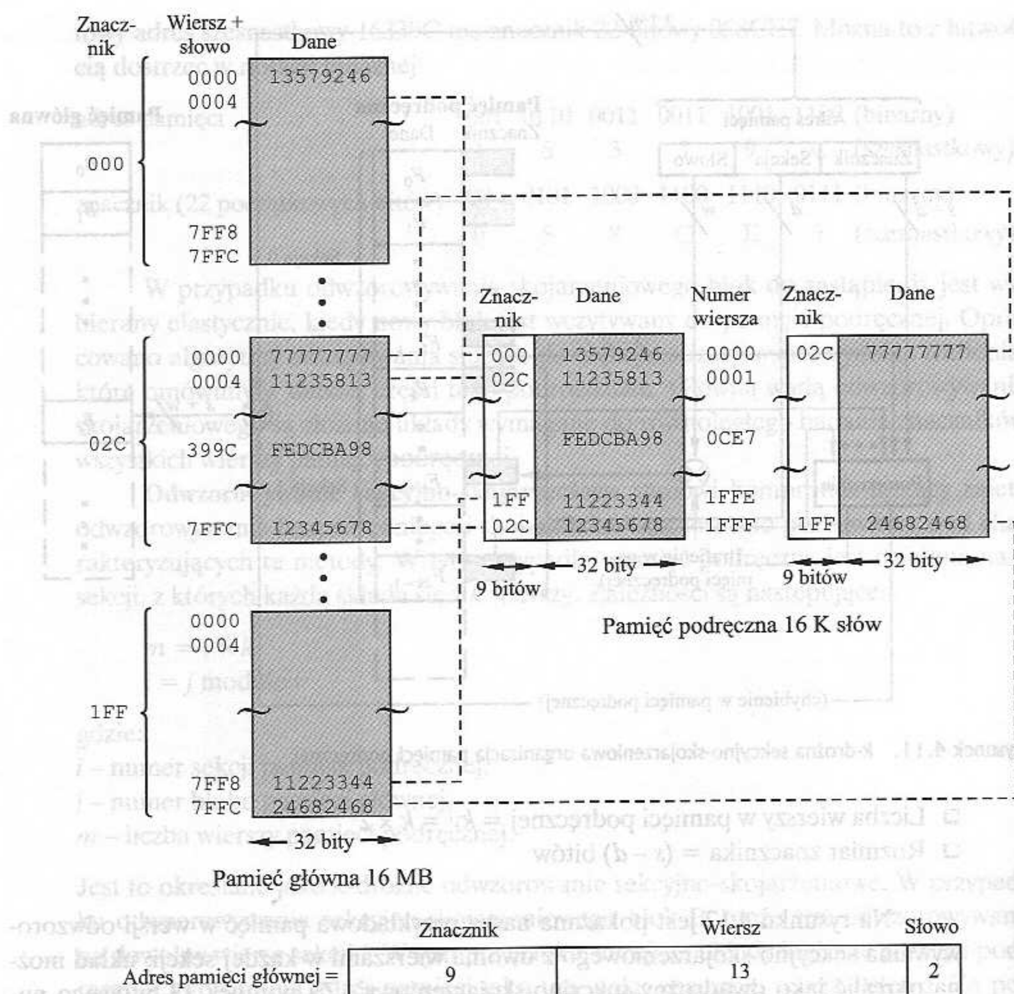
Rysunek 4.11. *k*-drożna sekcyjno-skojarzeniowa organizacja pamięci podręcznej

- Liczba wierszy w pamięci podręcznej = $kv = k \times 2^d$
- Rozmiar znacznika = $(s - d)$ bitów

Na rysunku 4.12 jest pokazana nasza przykładowa pamięć w wersji odwzorowywania sekcyjno-skojarzeniowego z dwoma wierszami w każdej sekcji; układ można określić jako dwudrożny sekcyjno-skojarzeniowy[5]. Za pomocą 13-bitowego numeru sekcji identyfikuje się jednoznacznie określoną sekcję złożoną z 2 wierszy wewnątrz pamięci skojarzeniowej. Określa się także numer bloku w pamięci głównej, modulo 2^{13}, co umożliwia odwzorowanie bloków na wierszach. Tak więc bloki 000000, 008000, 008000, ..., FF8000 pamięci głównej są przypisane sekcji 0 pamięci podręcznej. Dowolny z tych bloków może być załadowany do jednego z dwóch wierszy tej sekcji. Zauważmy, że żaden z dwóch bloków przypisanych tej samej sekcji pamięci podręcznej nie ma takiego samego numeru znacznika. W przypadku operacji odczytu 13-bitowy numer sekcji jest wykorzystywany do określenia, która sekcja ma być badana. Następnie jest sprawdzana zgodność obu wierszy sekcji z numerem znacznika adresu, do którego chcemy uzyskać dostęp.

W krańcowym przypadku, gdy $v = m$, $k = 1$, metoda sekcyjno-skojarzeniowa redukuje się do odwzorowywania bezpośredniego, a przy $v = 1$, $k = m$ – redukuje się

[5] Na rysunku 4.12 9-bitowy znacznik jest reprezentowany przez 3-cyfrową liczbę szesnastkową. Najbardziej znacząca cyfra szesnastkowa ma w rzeczywistości długość zaledwie 1 bitu.



Rysunek 4.12. Przykład dwudrożnego odwzorowania sekcyjno-skojarzeniowego

do odwzorowywania skojarzeniowego. Wykorzystywanie dwóch wierszy na sekcję ($v = m/2$, $k = 2$) jest najbardziej powszechną organizacją sekcyjno-skojarzeniową. Poprawia ona znacząco współczynnik trafienia w stosunku do odwzorowywania bezpośredniego. Czterodrożne odwzorowywanie sekcyjno-skojarzeniowe ($v = m/4$, $k = 4$) umożliwia uzyskanie umiarkowanej poprawy w zamian za względnie mały koszt dodatkowy [MAYB84, HILL89]. Dalsze zwiększanie liczby wierszy w sekcji przynosi małe efekty.

Algorytmy zastępowania

Gdy do pamięci podręcznej jest wprowadzany nowy blok, jeden z istniejących bloków musi być zastąpiony. W przypadku odwzorowywania bezpośredniego istnieje tylko jeden możliwy wiersz dla każdego określonego bloku i wybór nie jest możli-

wy. W przypadku metody skojarzeniowej i sekcyjno-skojarzeniowej wymagany jest algorytm zastępowania. Aby uzyskać dużą szybkość, algorytm taki musi być wbudowany w postaci sprzętowej. Wypróbowano wiele algorytmów. Wspomnimy o czterech najbardziej powszechnych. Prawdopodobnie najbardziej efektywny jest algorytm „najdawniej używany" (*least-recently used* – LRU). Algorytm określa, że należy zastąpić ten blok w sekcji, który pozostawał w pamięci podręcznej najdłużej bez odwoływania się do niego. W przypadku dwudrożnego odwzorowywania sekcyjno-skojarzeniowego jest to łatwe do wdrożenia. Każdy wiersz zawiera bit wykorzystania (USE). Gdy wiersz jest adresowany, jego bit USE jest ustawiany na 1, a bit USE pozostałego wiersza w tej sekcji jest ustawiany na 0. Jeśli blok ma być wczytany do sekcji, wykorzystuje się wiersz, którego bit USE ma wartość 0. Ponieważ zakładamy, że niedawno wykorzystywane lokacje pamięci są bardziej prawdopodobne do wykorzystania w przyszłości, LRU powinien dawać najlepszy współczynnik trafienia. Inną możliwość stwarza algorytm „pierwszy wchodzi – pierwszy wychodzi" (*first-in-first-out* – FIFO). Polega on na zastępowaniu tego bloku w sekcji, który najdłużej pozostawał w pamięci podręcznej. Algorytm FIFO jest łatwy do wdrożenia w postaci metody cyklicznego buforowania. Jeszcze inną możliwość stanowi algorytm „najrzadziej używany" (*least frequently med* – LFU). Określa on, że zastępowany jest ten blok w sekcji, którego dotyczyło najmniej odniesień. Algorytm LFU można wdrożyć przez skojarzenie licznika z każdym wierszem. Metodą abstrahującą od ilości odniesień jest przypadkowy wybór wśród kandydujących wierszy. Badania symulacyjne wykazały, że losowe zastępowanie prowadzi do niewiele tylko mniejszej wydajności w stosunku do algorytmów opartych na wykorzystywaniu [SMIT82].

Algorytm zapisu

Zanim blok pozostający w pamięci podręcznej będzie mógł być zastąpiony, konieczne jest rozważenie, czy musi on być zmieniony w pamięci podręcznej, a nie w pamięci głównej. Jeśli nie musi, to stary blok w pamięci podręcznej może być nadpisany. Jeśli musi, to znaczy, że na słowie zawartym w tej linii pamięci podręcznej musi być przeprowadzona przynajmniej jedna operacja zapisu, a pamięć główna musi być odpowiednio zaktualizowana. Możliwe są różne algorytmy zapisu wykorzystujące współzależności dotyczące wydajności i ekonomii. Istnieją dwa problemy, z którymi musimy się borykać. Po pierwsze, do pamięci głównej może mieć dostęp więcej niż jedno urządzenie. Na przykład moduł wejścia-wyjścia może być zdolny do odczytu--zapisu bezpośrednio w pamięci. Jeśli słowo byłoby zmienione tylko w pamięci podręcznej, to odpowiednie słowo w pamięci głównej byłoby nieaktualne. Ponadto, jeśli moduł wejścia-wyjścia zmieniłby słowo w pamięci głównej, to odpowiednie słowo w pamięci podręcznej byłoby nieważne. Bardziej złożony problem występuje, gdy mamy do czynienia z wieloma procesorami podłączonymi do tej samej magistrali i wyposażonymi we własne, lokalne pamięci podręczne. Jeśli w takiej sytuacji byłoby zmienione słowo w jednej pamięci podręcznej, spowodowałoby to unieważnienie słowa w innych pamięciach podręcznych.

Najprostsza metoda jest określana jako *zapis jednoczesny* (*write through*). Przy jej zastosowaniu wszystkie operacje zapisu są prowadzone jednocześnie zarówno do pamięci głównej, jak i do pamięci podręcznej, co zapewnia stałą aktualność danych w pamięci głównej. Jakikolwiek inny moduł procesor-pamięć podręczna może monitorować przesyłanie do pamięci głównej w celu zachowania *spójności* (*consistency*) swojej pamięci podręcznej. Główną wadą tej metody jest to, że generuje ona znaczny przepływ danych do pamięci i może przez to spowodować występowanie wąskich gardeł. Metoda alternatywna, nazwana metodą *zapisu opóźnionego* (*write back*), minimalizuje ilość operacji zapisu do pamięci. W przypadku tej metody aktualizuje się tylko pamięć podręczną. Gdy następuje taka aktualizacja, określana jest wartość bitu aktualizacji (UPDATE) skojarzonego z wierszem pamięci podręcznej. Jeśli następnie blok jest zastępowany, to podlega on wpisaniu do pamięci głównej tylko wtedy, gdy jest ustanowiony bit UPDATE. Problemem tej metody jest to, że część zawartości pamięci głównej jest nieaktualna, przez co dostęp modułów wejścia-wyjścia jest możliwy tylko za pośrednictwem pamięci podręcznej. Komplikuje to układy i stwarza możliwość wąskich gardeł. Doświadczenie wykazało, że udział zapisów w operacjach dostępu do pamięci jest rzędu 15% [SMIT82]. Jednak w zastosowaniach HPC liczba ta może sięgać 33% (mnożenie wektorów przez wektory), a nawet 50% (transpozycja macierzy).

W przypadku organizacji magistralowej, w której więcej niż jedno urządzenie (zwykle procesor) ma pamięć podręczną, a pamięć główna jest wspólna, powstaje nowy problem. Jeśli dane w jednej pamięci podręcznej są zmieniane, to operacja taka powoduje unieważnienie odpowiedniego słowa nie tylko w pamięci głównej, ale także w innych pamięciach podręcznych (jeśli to słowo w nich akurat występuje). Jeśli nawet jest stosowana metoda zapisu jednoczesnego, to inne pamięci podręczne mogą zawierać nieaktualne dane. System, który zapobiega temu problemowi, jest nazywany systemem zapewniającym spójność (*coherency*) pamięci podręcznej. Do możliwych rozwiązań problemu spójności pamięci podręcznej należą:

- **Obserwowanie magistrali z zapisem jednoczesnym.** Każdy sterownik pamięci podręcznej monitoruje linie adresowe w celu wykrycia operacji zapisu do pamięci dokonywanych przez inne jednostki nadrzędne magistrali. Jeśli inna jednostka nadrzędna wpisuje dane do takiej lokacji w pamięci wspólnej, która występuje również w pamięci podręcznej, to sterownik pamięci podręcznej unieważnia dostęp do pamięci podręcznej. Strategia ta jest uzależniona od tego, czy wszystkie sterowniki pamięci podręcznych posługują się metodą zapisu jednoczesnego.
- **Sprzętowe zapewnianie przezroczystości** (*hardware transparency*). Wykorzystywane są dodatkowe rozwiązania sprzętowe zapewniające, że wszystkie aktualizacje pamięci głównej dokonywane za pośrednictwem pamięci podręcznej znajdują odzwierciedlenie we wszystkich pamięciach podręcznych. Jeśli jeden procesor modyfikuje słowo w swojej pamięci podręcznej, aktualizacja ta jest wprowadzana również do pamięci głównej. Ponadto, wszystkie odpowiednie słowa w pozostałych pamięciach podręcznych są podobnie aktualizowane.

- **Pamięć wyłączona ze współpracy z pamięcią podręczną** (*non-cachable memory*). Tylko część pamięci głównej jest wspólna dla więcej niż jednego procesora i ta właśnie część jest oznaczana jako wyłączona ze współpracy z pamięcią podręczną. W takim systemie wszystkie operacje dostępu do pamięci wspólnej nie dotyczą pamięci podręcznych, ponieważ pamięć wspólna nigdy nie jest kopiowana do pamięci podręcznych. Pamięć wyłączona ze współpracy z pamięcią podręczną może być identyfikowana za pomocą układów logicznych wyboru mikroukładu lub najbardziej znaczących bitów adresu.

Przezroczystość pamięci podręcznej stanowi czynne pole badań. Temat ten zostanie rozwinięty w rozdziale 18.

Rozmiar wiersza

Kolejnym elementem projektowania jest rozmiar wiersza. Gdy blok danych jest pobierany i umieszczany w pamięci podręcznej, pobierane jest nie tylko żądane słowo, lecz również pewna liczba sąsiednich słów. Gdy zwiększamy rozmiar bloku od bardzo małego do dużego, współczynnik trafienia najpierw wzrasta ze względu na zasadę lokalności: wysokie prawdopodobieństwo, że dane sąsiadujące z wywołanym słowem będą potrzebne w niedalekiej przyszłości. W miarę wzrostu rozmiaru bloku do pamięci podręcznej jest doprowadzana coraz większa ilość użytecznych danych. Przy dalszym wzroście współczynnik trafienia zacznie jednak maleć, a prawdopodobieństwo wykorzystania nowo pobieranych informacji stanie się mniejsze niż prawdopodobieństwo ponownego użycia informacji, która jest zastępowana. Grają tu rolę dwa szczególne zjawiska:

- Użycie większych bloków powoduje zmniejszenie liczby bloków, które mieszczą się w pamięci podręcznej. Ponieważ każde wpisanie bloku powoduje usunięcie starej zawartości pamięci podręcznej, mała liczba bloków sprawia, że dane są usuwane niedługo po ich pobraniu.
- Gdy blok staje się większy, każde dodatkowe słowo jest dalsze od potrzebnego słowa, staje się więc mniej prawdopodobne jego użycie w niedalekiej przyszłości.

Związek między rozmiarem bloku a współczynnikiem trafienia jest złożony i uzależniony od stopnia lokalności określonego programu. Nie znaleziono definitywnej wartości optymalnej. Rozmiar od 8 do 32 bajtów wydaje się leżeć rozsądnie blisko optimum [SMIT87a, PRZY88, PRZY90, HAND98]. W systemach HPC najczęściej używane rozmiary wierszy pamięci podręcznych to 64 i 128 bajtów.

Liczba pamięci podręcznych

Kiedy wprowadzono pamięci podręczne, typowy system miał pojedynczą pamięć podręczną. Ostatnio stało się normą wykorzystywanie wielu pamięci podręcznych. Dwoma aspektami tego zagadnienia projektowego są: liczba poziomów pamięci oraz podział lub scalanie pamięci podręcznych.

Pamięci podręczne wielopoziomowe

W miarę wzrostu gęstości upakowania układów logicznych stało się możliwe dysponowanie pamięcią podręczną wewnątrz tego samego mikroukładu co procesor: określamy to mianem pamięci podręcznej w procesorze (*on-chip cache*). W porównaniu z pamięcią osiągalną za pomocą zewnętrznej magistrali wewnątrzprocesorowa pamięć podręczna umożliwia zmniejszenie aktywności procesora w magistrali zewnętrznej, przez to skraca się czasy wykonywania operacji i zwiększa się ogólną wydajność systemu. Jeśli wymagane rozkazy lub dane znajdują się w wewnątrzprocesorowej pamięci podręcznej, konieczność dostępu do magistrali jest eliminowana. Ze względu na krótsze ścieżki danych wewnątrz procesora w porównaniu z długościami magistrali, dostęp do wbudowanej pamięci podręcznej jest wyraźnie szybszy w porównaniu z cyklem magistrali, nawet w stanie nie wymagającym oczekiwania. Ponadto, w tym okresie magistrala pozostaje wolna i mogą zachodzić inne transfery.

Wbudowanie pamięci podręcznej do procesora pozostawia otwarte pytanie, czy nadal jest pożądana zewnętrzna (poza mikroukładem procesora) pamięć podręczna. Odpowiedź brzmi zwykle: tak. Większość współczesnych projektów zawiera zarówno pamięć podręczną wewnątrzprocesorową, jak i zewnętrzną. Wynikająca stąd organizacja jest znana jako dwupoziomowa pamięć podręczna, przy czym pamięć wewnętrzna jest oznaczana jako poziom 1 (L1), a zewnętrzna jako poziom 2 (L2). Przyczyna wprowadzania pamięci L2 jest następująca. Jeśli nie byłoby pamięci podręcznej L2, a procesor zgłosiłby zapotrzebowanie na dostęp do lokacji pamięci nie występującej w pamięci podręcznej L1, to procesor musiałby sięgać do pamięci DRAM lub ROM poprzez magistralę. Ze względu na typowo niewielką szybkość magistrali oraz stosunkowo długi czas dostępu do pamięci, rezultatem będzie zmniejszona wydajność. Z drugiej strony, jeśli zastosowano pamięć podręczną L2 SRAM (statyczna RAM), to często możliwe jest szybkie pobranie brakującej informacji. Jeśli pamięć SRAM jest dostatecznie szybka, aby dostosować się do szybkości magistrali, to możliwy jest dostęp do danych za pomocą transakcji o zerowym czasie oczekiwania, co stanowi najszybszy rodzaj transferów magistralowych.

Dwie właściwości współczesnych, wielopoziomowych pamięci podręcznych są warte podkreślenia. Po pierwsze, w przypadku pamięci podręcznej L2 znajdującej się poza strukturą procesora, w wielu rozwiązaniach jako ścieżki transferu między pamięcią L2 a procesorem nie używa się magistrali systemowej, lecz odrębnej ścieżki danych, co ma na celu ograniczenie obciążenia magistrali systemowej. Po drugie, dzięki ciągłemu zmniejszaniu rozmiarów elementów procesora, w wielu procesorach pamięć podręczna L2 jest wbudowana w tym samym mikroukładzie, co zwiększa wydajność.

Potencjalne oszczędności wynikające z zastosowania pamięci podręcznej L2 zależą od współczynników trafienia zarówno pamięci L1, jak i L2. W kilku badaniach wykazano, że na ogół zastosowanie pamięci podręcznej drugiego poziomu zwiększa wydajność (patrz np. [AZIM92], [NOVI93], [HAND98]). Jednak stosowanie pamięci wielopoziomowych komplikuje wszystkie problemy projektowe związane z pamięciami podręcznymi, w tym rozmiar, algorytm zastępowania i algorytm zapisu; zagadnienia te zostały przedstawione w [HAND98] i [PEIR99].

Jednolita a podzielona pamięć podręczna

Gdy po raz pierwszy wprowadzono wewnątrzprocesorową pamięć podręczną, w wielu projektach zawierających pojedynczą pamięć podręczną wykorzystywano ją zarówno do przechowywania danych, jak i rozkazów. Obecnie stało się powszechne dzielenie pamięci podręcznej na dwie: jedną przeznaczoną na rozkazy, drugą zaś na dane.

Istnieją dwie potencjalne zalety pamięci jednolitej:

- Dla danego rozmiaru pamięci podręcznej pamięć jednolita wyróżnia się większym współczynnikiem trafienia niż pamięć podzielona, ponieważ automatycznie równoważy ona pobieranie rozkazów z pobieraniem danych. To znaczy, jeśli struktura rozkazów przewiduje o wiele więcej pobrań rozkazów niż danych, pamięć podręczna będzie wykazywała tendencję do wypełniania się rozkazami. Jeżeli natomiast wymagane jest częstsze pobieranie danych, pamięć ta będzie się wypełniała danymi.
- Tylko jedna pamięć podręczna musi być zaprojektowana i wdrożona.

Mimo tych zalet występuje tendencja do stosowania pamięci podzielonych, szczególnie w przypadku maszyn superskalarnych, jak Pentium i PowerPC, co akcentuje równoległe wykonywanie rozkazów i wstępne pobieranie rozkazów przewidywanych do wykonywania. Kluczową zaletą podzielonej pamięci podręcznej jest to, że eliminowana jest rywalizacja o pamięć podręczną między procesorem rozkazów a jednostką wykonującą. Jest to ważne w każdym rozwiązaniu wykorzystującym potokowe przetwarzanie rozkazów. Typowo procesor pobiera rozkazy z wyprzedzeniem i napełnia bufor – rejestr potokowy – rozkazami, które mają być wykonane. Załóżmy teraz, że mamy do dyspozycji jednolitą pamięć podręczną dla rozkazów i danych. Gdy jednostka wykonująca sięga do pamięci w celu załadowania i zapisania danych, zapotrzebowanie jest doprowadzane do jednolitej pamięci podręcznej. Jeśli w tym samym czasie układ wyprzedzającego pobierania rozkazów wysyła zapotrzebowanie na odczytanie rozkazu z pamięci podręcznej, to zapotrzebowanie to zostanie czasowo zablokowane, żeby pamięć podręczna mogła najpierw obsłużyć jednostkę wykonującą, pozwalając jej na zakończenie właśnie wykonywanego rozkazu. Ta rywalizacja o pamięć podręczną może spowodować zmniejszenie wydajności, zakłócając wydajne wykorzystanie potoku rozkazów. Struktura podzielonej pamięci podręcznej umożliwia pokonanie tej trudności.

4.4. Organizacja pamięci podręcznej w Pentium 4 i PowerPC

Organizacja pamięci podręcznej w Pentium 4

Ewolucja organizacji pamięci podręcznej może być wyraźnie dostrzeżona w ewolucji mikroprocesorów Intela. Procesor 80386 nie zawiera wewnątrzprocesorowej pamięci podręcznej, natomiast 80486 zawiera wewnątrzprocesorową pamięć podręczną 8 KB, wykorzystującą wiersze o rozmiarze 16 bajtów i czterodrożną orga-

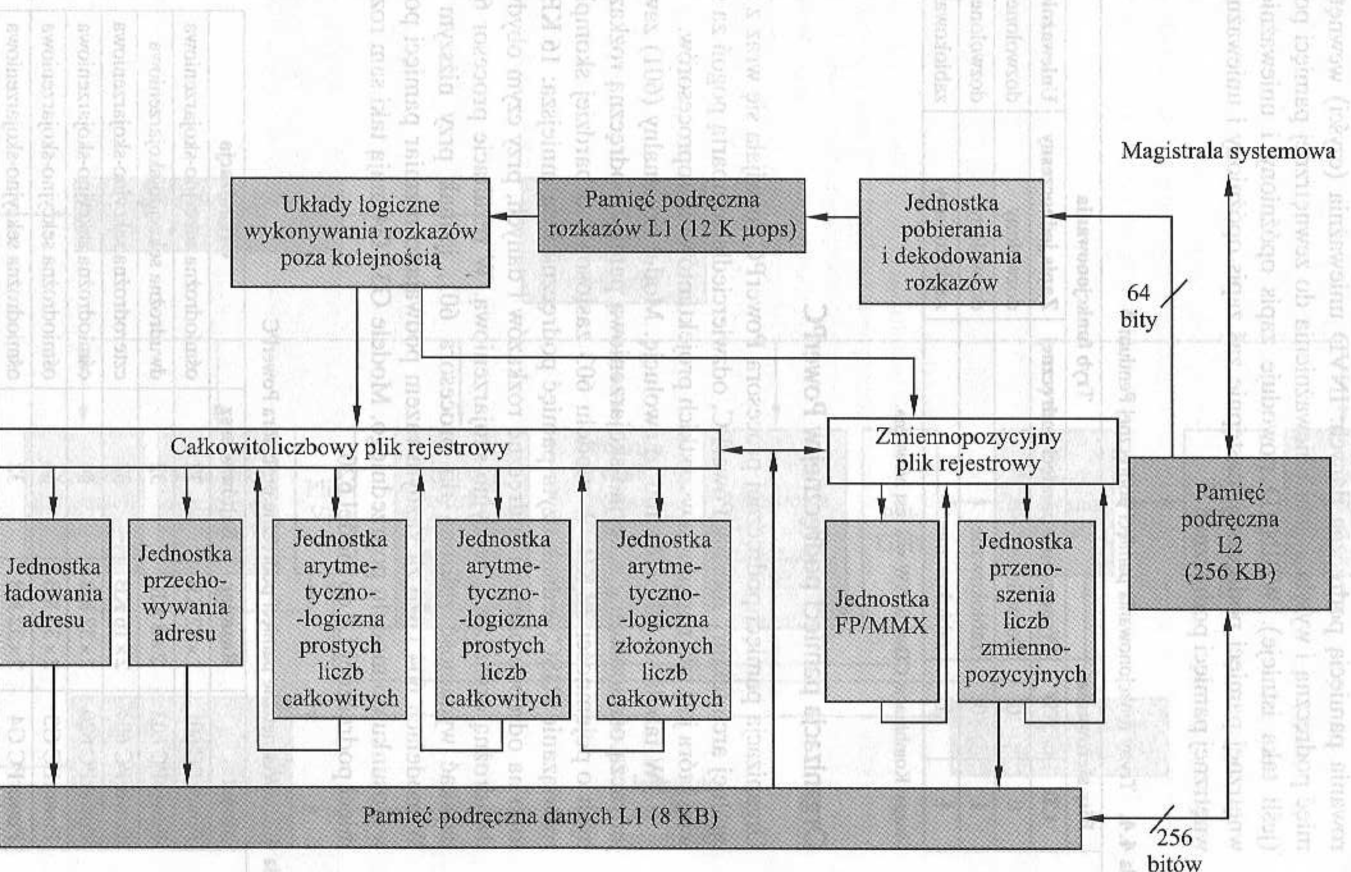
nizację sekcyjno-skojarzeniową. Wszystkie procesory Pentium zawierają dwie wewnątrzprocesorowe pamięci podręczne L1: jedną dla danych, drugą dla rozkazów. W Pentium 4 pamięć podręczna danych L1 ma rozmiar 8 KB, rozmiar wiersza wynosi w niej 64 bajty, organizacja zaś jest czterodrożna sekcyjno-skojarzeniowa. Pamięć podręczna rozkazów Pentium 4 zostanie przedstawiona później. Pentium 4 zawiera również pamięć podręczną L2 zasilającą obie pamięci L1. Pamięć L2 ma ośmiodrożną organizację sekcyjno-skojarzeniową o rozmiarze 256 KB, a rozmiar wiersza wynosi 128 bajtów.

Na rysunku 4.13 przedstawiono uproszczony obraz organizacji Pentium 4 ze szczególnym uwzględnieniem lokalizacji trzech pamięci podręcznych. Rdzeń procesora składa się z czterech głównych części:

- **Jednostka pobierania i dekodowania.** Pobiera rozkazy programu kolejno z pamięci podręcznej L2, dekoduje je na szereg mikrooperacji i zapisuje wyniki w pamięci podręcznej rozkazów L1.
- **Układy logiczne wykonywania rozkazów poza kolejnością.** Szeregowanie wykonywania mikrooperacji podlega wpływowi zależności danych i dostępności zasobów; dlatego wykonywanie mikrooperacji może być szeregowane w kolejności innej, niż wynika z ich pobierania ze strumienia rozkazów. W miarę jak pozwala na to czas, jednostka ta szereguje spekulatywne wykonywanie mikrooperacji, które mogą być potrzebne w przyszłości.
- **Jednostki wykonujące.** Wykonują one mikrooperacje, pobierając wymagane dane z pamięci podręcznej danych L1 i tymczasowo zapisując wyniki w rejestrach.
- **Podsystem pamięci.** Jednostka ta obejmuje pamięć podręczną L2 i magistralę systemową, która umożliwia dostęp do pamięci głównej, gdy następuje chybienie w pamięciach podręcznych L1 i L2, a także do zasobów wejścia-wyjścia systemu.

W odróżnieniu od organizacji wszystkich poprzednich modeli Pentium, a także większości pozostałych procesorów, pamięć podręczna rozkazów Pentium 4 jest umiejscowiona między układami logicznymi dekodowania rozkazów a rdzeniem wykonywania rozkazów. Rozumowanie kryjące się za takim rozwiązaniem jest następujące. Jak zostanie to przedstawione w rozdziale 14, Pentium dekoduje – lub tłumaczy – rozkazy maszynowe Pentium na proste rozkazy w stylu RISC zwane mikrooperacjami. Użycie prostych mikrooperacji o ustalonej długości umożliwia zastosowanie superskalarnego przetwarzania potokowego i technik szeregowania zwiększających wydajność. Jednak rozkazy maszynowe Pentium są niewygodne do dekodowania; mają zmienne liczby bajtów i wiele różnych opcji. Okazuje się, że wydajność ulega poprawie, jeśli to dekodowanie jest realizowane niezależnie od układów logicznych szeregowania i przetwarzania potokowego. Powrócimy do tego tematu w rozdziale 14.

W pamięci podręcznej danych stosuje się algorytm zapisu opóźnionego: dane są zapisywane w pamięci głównej tylko wówczas, gdy są usuwane z pamięci podręcznej i nastąpiła ich aktualizacja. Procesor Pentium 4 może być dynamicznie konfigurowany tak, aby obsługiwał zapis jednoczesny.



Rysunek 4.13. Schemat blokowy Pentium 4

Pamięć podręczna danych L1 jest sterowana za pomocą dwóch bitów w jednym z rejestrów sterowania, noszących etykiety CD (*cache disable*, blokowanie pamięci podręcznej) i NW (*not write-through*, wyłączenie zapisu jednoczesnego) – zobacz tabela 4.4. Istnieją również dwa rozkazy Pentium 4, które mogą służyć do sterowania pamięcią podręczną danych: INVD unieważnia (czyści) wewnętrzną pamięć podręczną i wysyła sygnał unieważnienia do zewnętrznej pamięci podręcznej (jeśli taka istnieje). WBINVD powoduje zapis opóźniony i unieważnienie wewnętrznej pamięci podręcznej, następnie zaś zapis opóźniony i unieważnienie zewnętrznej pamięci podręcznej.

Tabela 4.4. Tryby funkcjonowania pamięci podręcznej Pentium 4

Bity sterowania		Tryb funkcjonowania		
CD	NW	Wypełnianie pamięci podręcznej	Zapis jednoczesny	Unieważnienie
0	0	dozwolone	dozwolony	dozwolone
1	0	zablokowane	dozwolony	dozwolone
1	1	zablokowane	zablokowany	zablokowane

Uwaga: Kombinacja CD = 0, NW = 1 jest nieważna.

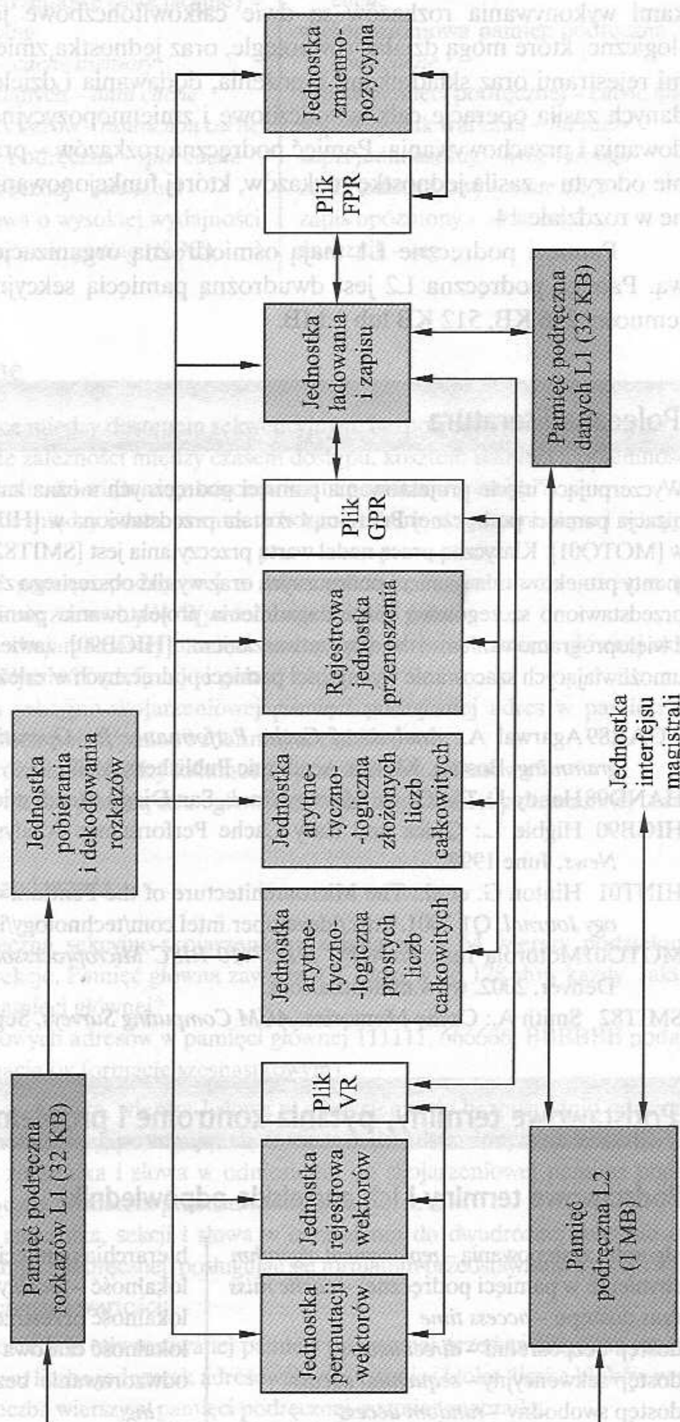
Organizacja pamięci podręcznej w PowerPC

Organizacja pamięci podręcznej procesora PowerPC rozwijała się wraz z ewolucją ogólnej architektury rodziny PowerPC, odzwierciedlając upartą pogoń za wydajnością, która jest siłą napędową wszystkich projektantów mikroprocesorów.

W tabeli 4.5 zilustrowano tę ewolucję. Model oryginalny (601) zawiera pojedynczą ośmiodrożną, sekcyjno-skojarzeniową pamięć podręczną rozkazów i danych o pojemności 32 KB. W modelu 603 zastosowano bardziej skomplikowane rozwiązanie RISC, jednak użyta pamięć podręczna była mniejsza: 16 KB podzielono na odrębne pamięci podręczne rozkazów i danych, przy czym obydwie mają dwudrożną organizację sekcyjno-skojarzeniową. W rezultacie procesor 603 może uzyskać wydajność zbliżoną do procesora 601, jednak przy niższym koszcie. W modelach 604 i 620 za każdym razem podwajano rozmiar pamięci podręcznej w stosunku do modelu poprzedniego. Modele G3 i G4 mają taki sam rozmiar pamięci podręcznej L1 co model 620.

Tabela 4.7. Wewnętrzne pamięci podręczne procesora PowerPC

Model	Rozmiar	Bajtów/wiersz	Organizacja
PowerPC 601	1×32 KB	32	ośmiodrożna sekcyjno-skojarzeniowa
PowerPC 603	2×8 KB	32	dwudrożna sekcyjno-skojarzeniowa
PowerPC 604	2×16 KB	32	czterodrożna sekcyjno-skojarzeniowa
PowerPC 620	2×32 KB	64	ośmiodrożna sekcyjno-skojarzeniowa
PowerPC G3	2×32 KB	64	ośmiodrożna sekcyjno-skojarzeniowa
PowerPC G4	2×32 KB	32	ośmiodrożna sekcyjno-skojarzeniowa



Rysunek 4.14. Schemat blokowy procesora PowerPC

Na rysunku 4.14 przedstawiono uproszczony schemat organizacji PowerPC G4, uwydatniając rozmieszczenie obydwu pamięci podręcznych. Rdzeniowymi jednostkami wykonywania rozkazów są dwie całkowitoliczbowe jednostki arytmetyczno-logiczne, które mogą działać równolegle, oraz jednostka zmiennopozycyjna z własnymi rejestrami oraz składnikami mnożenia, dodawania i dzielenia. Pamięć podręczna danych zasila operacje całkowitoliczbowe i zmiennopozycyjne poprzez jednostkę ładowania i przechowywania. Pamięć podręczna rozkazów – pracująca wyłącznie w trybie odczytu – zasila jednostkę rozkazów, której funkcjonowanie zostanie przedstawione w rozdziale 14.

Pamięci podręczne L1 mają ośmiodrożną organizację sekcyjno-skojarzeniową. Pamięć podręczna L2 jest dwudrożną pamięcią sekcyjno-skojarzeniową o pojemności 256 KB, 512 KB lub 1 MB.

4.5. Polecana literatura

Wyczerpujące ujęcie projektowania pamięci podręcznych można znaleźć w [HAND98]. Organizacja pamięci podręcznej Pentium 4 została przedstawiona w [HINT01], zaś PowerPC G4 – w [MOTO01]. Klasyczną pracą nadal wartą przeczytania jest [SMIT82]; obejmuje ona różne elementy projektowania pamięci podręcznych oraz wyniki obszernego zbioru analiz. W [AGAR89] przedstawiono szczegółowo różne zagadnienia projektowania pamięci podręcznych związane z wieloprogramowaniem i wieloprzetwarzaniem. [HIGB90] zawiera zbiór prostych formuł umożliwiających szacowanie wydajności pamięci podręcznych w zależności od ich parametrów.

AGAR89 Agarwal A.: *Analysis of Cache Performance for Operating Systems and Multiprogramming*. Boston, Kluwer Academic Publishers, 1989.

HAND98 Handy J.: *The Cache Memory Book*. San Diego, Academic Press, 1993.

HIGB90 Higbie L.: Quick and Easy Cache Performance Analysis. *Computer Architecture News*, June 1990.

HINT01 Hinton G. et al.: The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal*, Q1 2001. http://developer.intel.com/technology/itj

MOTO01 Motorola Inc.: *PowerPC MPC7410 RISC Microprocessor Hardware Specifications*. Denver, 2002. www.motorola.com.

SMIT82 Smith A.: Cache Memories. *ACM Computing Surveys*, September 1992.

4.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

algorytm zastępowania – *replacement algorithm*
chybienie w pamięci podręcznej – *cache miss*
czas dostępu – *access time*
dostęp bezpośredni – *direct access*
dostęp sekwencyjny – *sequential access*
dostęp swobodny – *random access*
hierarchia pamięci – *memory hierarchy*
lokalność – *locality*
lokalność przestrzenna – *spatial locality*
lokalność czasowa – *temporal locality*
odwzorowanie bezpośrednie – *direct mapping*

odwzorowanie sekcyjno-skojarzeniowe – *set associative mapping*
odwzorowanie skojarzeniowe (asocjacyjne) – *associative mapping*
pamięć podręczna – *cache memory*
pamięć podręczna danych – *data cache*
pamięć podręczna rozkazów – *instruction cache*
rozdzielona pamięć podręczna – *split cache*
sekcja pamięci podręcznej – *cache set*
technika obliczeniowa o wysokiej wydajności – *high performance computing* (HPC)
trafienie w pamięci podręcznej – *cache hit*
ujednolicona pamięć podręczna – *unified cache*
wielopoziomowa pamięć podręczna – *multi-level cache*
wiersz pamięci podręcznej – *cache line*
współczynnik trafienia – *hit ratio*
zapis jednoczesny – *write through*
zapis jednorazowy – *write once*
zapis opóźniony – *write back*
znacznik – *tag*

Pytania kontrolne

4.1. Jakie są różnice między dostępem sekwencyjnym, bezpośrednim a swobodnym?
4.2. Jakie są ogólne zależności między czasem dostępu, kosztem pamięci a pojemnością?
4.3. Jak zasada lokalności wiąże się z użyciem wielu poziomów pamięci?
4.4. Jakie są różnice między odwzorowaniem bezpośrednim, skojarzeniowym i sekcyjno-skojarzeniowym?
4.5. W przypadku pamięci podręcznej o odwzorowaniu bezpośrednim adres w pamięci głównej składa się z trzech pól. Wymień i zdefiniuj te pola.
4.6. W przypadku skojarzeniowej pamięci podręcznej adres w pamięci głównej składa się z dwóch pól. Wymień i zdefiniuj te pola.
4.7. W przypadku sekcyjno-skojarzeniowej pamięci podręcznej adres w pamięci głównej składa się z trzech pól. Wymień i zdefiniuj te pola.
4.8. Jakie jest rozróżnienie między lokalnością przestrzenną a czasową?
4.9. Ogólnie rzecz biorąc, jakie są strategie wykorzystania lokalności przestrzennej i czasowej?

Problemy do rozwiązania

4.1. Pamięć podręczna sekcyjno-skojarzeniowa składa się z 64 wierszy podzielonych na 4-wierszowe sekcje. Pamięć główna zawiera 4 K bloków po 128 słów każdy. Jaki format mają adresy pamięci głównej?
4.2. Dla szesnastkowych adresów w pamięci głównej 111111, 666666, BBBBBB podaj następujące informacje (w formacie szesnastkowym):

(a) Wartości znacznika, wiersza i słowa w odniesieniu do bezpośrednio odwzorowanej pamięci podręcznej, posługując się formatem przedstawionym na rys. 4.8.

(b) Wartości znacznika i słowa w odniesieniu do skojarzeniowej pamięci podręcznej, posługując się formatem przedstawionym na rys. 4.10.

(c) Wartości znacznika, sekcji i słowa w odniesieniu do dwudrożnej sekcyjno-skojarzeniowej pamięci podręcznej, posługując się formatem przedstawionym na rys. 4.12.

4.3. Wymień następujące wartości:

(a) Dla bezpośrednio odwzorowanej pamięci podręcznej przedstawionej na rys. 4.8: długość adresu, liczba jednostek adresowalnych, rozmiar bloku, liczba bloków w pamięci głównej, liczba wierszy w pamięci podręcznej, rozmiar znacznika.

(b) Dla skojarzeniowej pamięci podręcznej przedstawionej na rys. 4.10: długość adresu, liczba jednostek adresowalnych, rozmiar bloku, liczba bloków w pamięci głównej, liczba wierszy w pamięci podręcznej, rozmiar znacznika.

(c) Dla dwudrożnej sekcyjno-skojarzeniowej pamięci podręcznej przedstawionej na rys. 4.12: długość adresu, liczba jednostek adresowalnych, rozmiar bloku, liczba bloków w pamięci głównej, liczba wierszy w sekcji, liczba sekcji, liczba wierszy w pamięci podręcznej, rozmiar znacznika.

4.4. Rozważ mikroprocesor 32-bitowy mający wewnętrzną, czterodrożną pamięć podręczną sekcyjno-skojarzeniową 16 KB. Załóż, że wiersz pamięci podręcznej ma długość czterech słów 32-bitowych. Narysuj schemat blokowy tej pamięci podręcznej ukazujący jej organizację oraz ujawniający sposób wykorzystania różnych pól adresowych do ustalania trafień i chybień. Gdzie w pamięci podręcznej jest odwzorowane słowo z lokacji pamięci ABCDE8F8?
Źródło: [ALEX93].

4.5. Dane są następujące własności zewnętrznej pamięci podręcznej: czterodrożna sekcyjno-skojarzeniowa, rozmiar linii – dwa słowa 16-bitowe, może przyjąć 4 K słów 32-bitowych z pamięci głównej, wykorzystywana z procesorem 16-bitowym, który generuje adresy 24-bitowe. Zaprojektuj strukturę pamięci podręcznej ze wszystkimi niezbędnymi informacjami i pokaż, jak interpretuje ona adresy z procesora.
Źródło: [ALEX93].

4.6. Intel 80486 ma wewnętrzną, zintegrowaną w mikroukładzie pamięć podręczną. Zawiera ona 8 KB danych, ma czterodrożną organizację sekcyjno-skojarzeniową, a długość bloku jest równa czterem słowom 32-bitowym. Pamięć ta jest zorganizowana w postaci 128 sekcji. Istnieje pojedynczy „bit ważności wiersza" oraz trzy bity „LRU" (B0, B1 i B2) przypisane każdemu wierszowi. W przypadku chybienia w pamięci podręcznej procesor 80486 odczytuje 16-bajtowy wiersz z pamięci głównej, kierując go do pakietu odczytu magistrali pamięci. Narysuj uproszczony schemat pamięci podręcznej i pokaż, jak są interpretowane różne pola adresu.
Źródło: [ALEX93].

4.7. Rozważmy komputer z bajtowo adresowalną pamięcią główną o pojemności 2^{16} bajtów i o rozmiarze bloków 8 bajtów. Załóżmy, że w tym komputerze została użyta bezpośrednio odwzorowana pamięć podręczna składająca się z 32 wierszy.

(a) W jaki sposób 16-bitowy adres pamięci jest podzielony na znacznik, numer wiersza i numer bajta?

(b) W którym wierszu zostaną zapisane bajty każdego spośród następujących adresów?
0001 0001 0001 1011
1100 0011 0011 0100
1101 0000 0001 1101
1010 1010 1010 1010

(c) Załóżmy, że w pamięci podręcznej jest zapisany bajt o adresie 0001 1010 0001 1010. Jakie są adresy pozostałych bajtów zapisanych razem z nim?

4.8. Algorytm zastępowania w procesorze Intel 80486 jest określany jako **pseudo-LRU**. Z każdą ze 128 sekcji zawierających po cztery wiersze (oznaczone L0, L1, L2 i L3) są skojarzone trzy bity: B0, B1 i B2. Algorytm wymiany działa następująco. Gdy wiersz musi być wymieniony, pamięć podręczna określa najpierw, czy ostatnie wykorzystanie dotyczyło L0, L1, L2 czy też L3. Następnie określa, która para bloków była najdawniej

wykorzystywana i znakuje ją jako przeznaczoną do wymiany. Schemat logiczny został przedstawiony na rys. 4.15.

(a) Podaj sposób ustawiania bitów B0, B1 i B2 oraz sposób ich wykorzystania w algorytmie wymiany.

(b) Pokaż, że algorytm wykorzystywany przez procesor 80486 stanowi przybliżenie prawdziwego algorytmu LRU. *Wskazówka*: rozważ przypadek, w którym kolejność ostatniego wykorzystania była następująca: L0, L2, L3, L1.

(c) Wykaż, że prawdziwy algorytm LRU wymaga 6 bitów na sekcję.



Rysunek 4.15. Algorytm zastępowania wewnętrznej pamięci podręcznej procesora Intel 80486

4.9. Blok podręcznej pamięci sekcyjno-skojarzeniowej ma rozmiar 4 słów 16-bitowych, a sekcja ma rozmiar 2 bloków. Pamięć ta może przyjąć 4048 słów. Pamięć główna współpracująca z tą pamięcią ma pojemność 64 K×32 bit. Zaprojektuj strukturę pamięci podręcznej i pokaż, jak są interpretowane adresy z procesora.
Źródło: [ALEX93].

4.10. Rozważmy system pamięciowy, w którym jest używany adres 32-bitowy do adresowania na poziomie bajtowym, oraz pamięć podręczną, w której użyto 64-bajtowego rozmiaru wiersza.

(a) Załóżmy, że pamięć podręczna jest odwzorowana bezpośrednio, a w 20-bitowym adresie znajduje się pole znacznika. Podaj format adresu i określ następujące parametry: liczba jednostek adresowalnych, liczba bloków w pamięci głównej, liczba wierszy w pamięci podręcznej, rozmiar znacznika.

(b) Załóżmy, że pamięć podręczna jest skojarzeniowa. Podaj format adresu i określ następujące parametry: liczba jednostek adresowalnych, liczba bloków w pamięci głównej, liczba wierszy w pamięci podręcznej, rozmiar znacznika.

(c) Załóżmy, że pamięć podręczna jest czterodrożna, sekcyjno-skojarzeniowa z polem znacznika w 9-bitowym adresie. Podaj format adresu i określ następujące parametry: liczba jednostek adresowalnych, liczba bloków w pamięci głównej, liczba wierszy w sekcji, liczba sekcji w pamięci podręcznej, liczba wierszy w pamięci podręcznej, rozmiar znacznika.

4.11. Opisz prostą technikę implementacji algorytmu zastępowania LRU w czterodrożnej sekcyjno-skojarzeniowej pamięci podręcznej.

4.12. Rozważmy następujący kod:

```
for (i = 0; i < 20; i++)
    for (j = 0; j < 10; j++)
        a[i] = a [i] * j
```

(a) Podaj jeden przykład lokalności przestrzennej w tym kodzie.
(b) Podaj jeden przykład lokalności czasowej w tym kodzie.

4.13. Uogólnij równania 4.1 i 4.2 z dodatku 4A tak, aby odnosiły się do N-poziomowych hierarchii pamięciowych.

4.14. System komputerowy zawiera pamięć główną o pojemności 32 K słów 16-bitowych. Ma także pamięć podręczną 4 K słów podzieloną na 4-wierszowe sekcje z 64 słowami w każdym wierszu. Załóż, że pamięć podręczna jest na początku pusta. Procesor pobiera słowa z lokacji 0, 1, 2, ..., 4351 w tym właśnie porządku. Następnie powtarza tę sekwencję pobierania jeszcze 9 razy. Pamięć podręczna jest 10 razy szybsza niż pamięć główna. Oszacuj korzyść wynikającą z zastosowania pamięci podręcznej. Przyjmij, że wymiana bloków jest zgodna z algorytmem LRU.

4.15. Rozważmy system pamięciowy o następujących parametrach:

$T_c = 100$ ns $\quad C_c = 0{,}01$ c/bit
$T_m = 1200$ ns $\quad C_m = 0{,}001$ c/bit

(a) Jaki jest koszt jednego MB pamięci głównej?
(b) Jaki jest koszt jednego MB pamięci głównej zbudowanej przy użyciu technologii pamięci podręcznej?
(c) Jeśli efektywny czas dostępu jest o 10% większy, niż czas dostępu pamięci podręcznej, jaki jest współczynnik trafienia H?

4.16. Komputer jest wyposażony w pamięć podręczną, pamięć główną i dysk służący do utworzenia pamięci wirtualnej. Jeśli poszukiwane słowo znajduje się w pamięci podręcznej, czas dostępu do niego wynosi 20 ns. Jeśli natomiast nie znajduje się w pamięci podręcznej, a w pamięci głównej, wymagane jest 60 ns, aby załadować je do pamięci podręcznej, po czym odniesienie do tego słowa rozpoczyna się od nowa. Jeśli słowa nie ma w pamięci głównej, potrzeba 12 ms na pobranie go z dysku, 60 ns na skopiowanie go do pamięci podręcznej, po czym odniesienie do tego słowa rozpoczyna się od nowa. Współczynnik trafienia pamięci podręcznej wynosi 0,9, a pamięci głównej – 0,6. Jaki jest średni czas w ns wymagany do tego, aby uzyskać dostęp do słowa w tym systemie?

Dodatek 4A. Charakterystyka wydajności pamięci dwupoziomowych

W rozdziale tym omówiliśmy pamięć podręczną działającą jako bufor między pamięcią główną a procesorem. Mamy tu do czynienia z dwupoziomową pamięcią wewnętrzną. Architektura taka umożliwia zwiększenie wydajności w stosunku do porównywalnej pamięci jednopoziomowej przez wykorzystanie właściwości zwanej lokalnością (tę właściwość przeanalizujemy poniżej).

Mechanizm pamięci podręcznej stanowi część architektury komputera zrealizowanej sprzętowo i jest zwykle niewidoczny dla systemu operacyjnego. Występują

ponadto dwa inne przypadki pamięci dwupoziomowych, które również wykorzystują lokalność i które – przynajmniej częściowo – są implementowane w systemie operacyjnym: pamięć wirtualna i dyskowa pamięć podręczna (tab. 4.6). Pamięć wirtualna jest opisana w rozdz. 8; dyskowa pamięć podręczna wykracza poza zakres tej książki, jest jednak przedstawiona w [STAL01]. W tym dodatku rozpatrzymy wybrane charakterystyki wydajnościowe pamięci dwupoziomowych, które są wspólne dla wszystkich trzech przypadków.

Tabela 4.6. Własności pamięci dwupoziomowych

	Pamięć podręczna pamięci głównej	**Pamięć wirtualna (stronicowana)**	**Dyskowa pamięć podręczna**
Typowy stosunek czasów dostępu	40/1 (z wbudowanej pamięci podręcznej do pamięci głównej) 10/1 (z zewnętrznej pamięci podręcznej do pamięci głównej)	10000/1 (z pamięci głównej na dysk)	10000/1 (z pamięci głównej na dysk)
System zarządzania pamięcią	wdrażany układowo	kombinacja układowego i programowego	wdrażany w postaci oprogramowania systemowego
Typowy rozmiar bloku	4÷128 B	64÷4096 B	64÷4096 B
Dostęp procesora do drugiego poziomu	bezpośredni	pośredni	pośredni

Lokalność

Podstawą zwiększonej wydajności pamięci dwupoziomowej jest zasada znana jako *lokalność odniesień* [DENN68]. Zgodnie z tą zasadą odniesienia do pamięci wykazują tendencję do grupowania się. W ciągu długiego czasu wykorzystywane ugrupowania zmieniają się, jednak w krótkim czasie procesor przede wszystkim pracuje z ustalonymi ugrupowaniami odniesień do pamięci.

Z intuicyjnego punktu widzenia zasada lokalności ma sens. Rozważmy następującą linię rozumowania:

1. Z wyjątkiem rozkazów skoku i wywołania, które stanowią małą część wszystkich rozkazów programu, realizacja programu ma charakter sekwencyjny. Tak więc, w większości przypadków następny rozkaz przewidziany do pobrania następuje bezpośrednio po ostatnio pobranym rozkazie.
2. Rzadkością jest występowanie długich, nieprzerwanych sekwencji wywołań procedury, po których następowałyby odpowiednie sekwencje powrotów. Program utrzymuje się raczej w wąskim zakresie głębokości wywołań procedury. Tak więc, w ciągu krótkiego czasu odwołania do rozkazów wykazują tendencję do pozostawania w obrębie niewielu procedur.
3. Większość tworów iteracyjnych składa się ze względnie małej liczby wielokrotnie powtarzanych rozkazów. Podczas iteracji obliczenia pozostają ograniczone do niewielkiej, zwartej części programu.

4. W wielu programach znaczna część obliczeń obejmuje przetwarzanie struktur danych, takich jak tablice lub szeregi rekordów. W wielu przypadkach kolejne odniesienia do tych struktur danych będą dotyczyły zbiorów położonych w sąsiedztwie.

Ten sposób rozumowania znalazł potwierdzenie w wielu badaniach. Rozpatrzmy na przykład punkt 1. Przeprowadzono wiele badań, których celem było przeanalizowanie działania programów wyrażonych w języku wysokiego poziomu. W tabeli 4.7 są przedstawione ich główne wyniki, z uwzględnieniem częstości występowania różnych typów instrukcji podczas wykonywania programów. Pochodzą one z niżej przedstawionych badań.

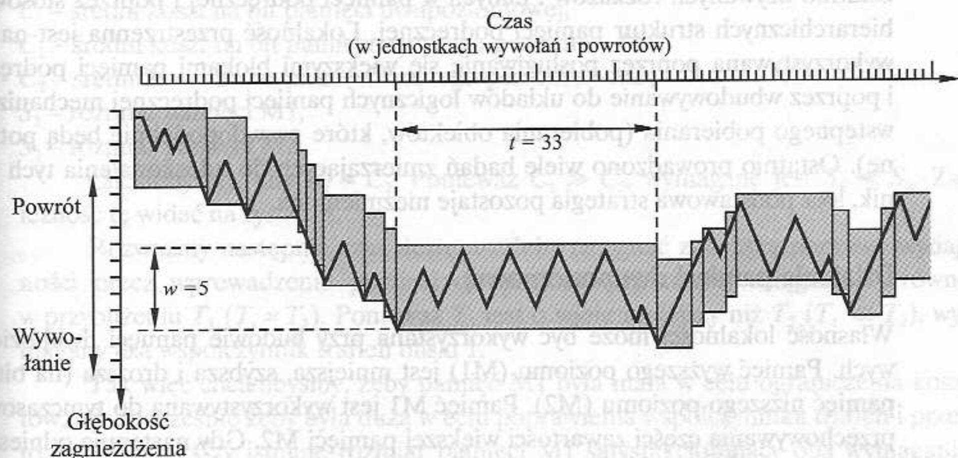
Tabela 4.7. Względna dynamiczna częstość występowania operacji języka wysokiego poziomu

Badanie	[HUCK83]	[KNUT71]	[PATT82]		[TANE78]
Język	Pascal	FORTRAN	Pascal	C	SAL
Zadania	naukowe	studenckie	systemowe	systemowe	systemowe
Przypisania	74	67	45	38	42
Pętle	4	3	5	3	4
Wywołania	1	3	15	12	12
IF	20	11	29	43	36
GO TO	2	9	–	3	–
Inne	–	7	6	1	6

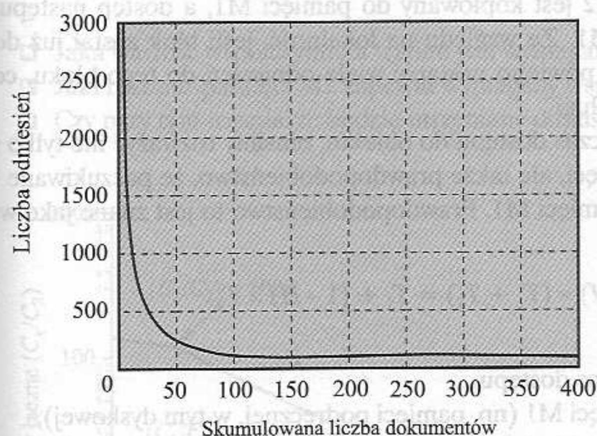
W najwcześniejszym badaniu funkcjonowania języków programowania, przeprowadzonym przez Knutha [KNUT71], przeanalizowano zbiór programów w FORTRAN-ie wykorzystywanych jako ćwiczenia dla studentów. Tanenbaum [TANE78] opublikował pomiary ponad 300 procedur wykorzystywanych w programach systemów operacyjnych, napisanych w języku umożliwiającym programowanie strukturalne (SAL). Patterson i Sequin [PATT82a] przeanalizowali zestaw pomiarów dokonanych na kompilatorach, programach do składania tekstu, CAD, sortowania i porównywania plików. Zbadano języki programowania C i Pascal. Huck [HUCK83] przeanalizował cztery programy reprezentujące zbiór obliczeń naukowych o ogólnym przeznaczeniu, w tym szybkie transformacje Fouriera i całkowanie układów równań różniczkowych. Wyniki tych badań prowadzą do zgodnego wniosku, że rozkazy skoku i wywołania stanowią tylko część instrukcji wykonywanych w czasie życia programu. Badania te stanowią więc potwierdzenie pierwszego z wyżej wymienionych twierdzeń.

Drugie twierdzenie znajduje uzasadnienie w badaniach przedstawionych w [PATT85a]. Jest to zilustrowane na rys. 4.16, na którym są pokazane procesy wywołanie-powrót. Każde wywołanie jest reprezentowane przez linię poprowadzoną w dół i na prawo, a każdy powrót przez linię poprowadzoną do góry i na prawo. Na rysunku zdefiniowano *okno* o głębokości 5. Jedynie sekwencja wywołań i powrotów powodująca efektywne przesunięcie w jakimkolwiek kierunku przekraczające 6 spra-

wia, że okno ulega przesunięciu. Jak można zauważyć, wykonywanie programu może pozostawać w ramach stacjonarnego okna przez całkiem długi czas. Badania tej samej grupy programów w C i Pascalu wykazały, że okno o głębokości 8 wymaga przesunięcia zaledwie w przypadku mniej niż 1% wywołań lub powrotów [TAMI83].



Rysunek 4.16. Przebieg wywołań i powrotów w programach



Rysunek 4.17. Lokalność odniesień do stron WWW [BAEN97]

Zasada lokalności odniesień jest nadal potwierdzana w nowszych badaniach. Na przykład na rys. 4.17 przedstawiono wyniki badań wzorów dostępu do strony WWW w pewnej witrynie.

W literaturze rozróżnia się lokalność przestrzenną i czasową. **Lokalność przestrzenna** odnosi się do tendencji do grupowania odniesień do miejsc w pamięci. Odzwierciedla to skłonność procesora do sekwencyjnego sięgania po rozkazy. Lokalność przestrzenna odzwierciedla również skłonność programów do sekwencyjnego sięgania do danych, co zachodzi np. przy przetwarzaniu tablic danych. **Lokalność**

czasowa oznacza skłonność procesora do sięgania do tych miejsc w pamięci, które były ostatnio wykorzystywane. Gdy na przykład jest wykonywana pętla iteracyjna, procesor powtarzalnie wykonuje taki sam zbiór rozkazów.

Tradycyjnie lokalność czasowa jest wykorzystywana poprzez utrzymywanie ostatnio używanych rozkazów i danych w pamięci podręcznej i poprzez stosowanie hierarchicznych struktur pamięci podręcznej. Lokalność przestrzenna jest na ogół wykorzystywana poprzez posługiwanie się większymi blokami pamięci podręcznej i poprzez wbudowywanie do układów logicznych pamięci podręcznej mechanizmów wstępnego pobierania (pobierania obiektów, które prawdopodobnie będą potrzebne). Ostatnio prowadzono wiele badań zmierzających do udoskonalenia tych technik, lecz podstawowa strategia pozostaje niezmieniona.

Działanie pamięci dwupoziomowej

Własność lokalności może być wykorzystana przy budowie pamięci dwupoziomowych. Pamięć wyższego poziomu (M1) jest mniejsza, szybsza i droższa (na bit) niż pamięć niższego poziomu (M2). Pamięć M1 jest wykorzystywana do tymczasowego przechowywania części zawartości większej pamięci M2. Gdy następuje odniesienie do pamięci, przeprowadzana jest próba znalezienia poszukiwanych danych w pamięci M1. Jeśli się ona powiedzie, wykonywany jest szybki dostęp. Jeśli nie, to odpowiedni blok pamięci M2 jest kopiowany do pamięci M1, a dostęp następuje za pośrednictwem pamięci M1. Ze względu na lokalność, jeśli blok został już doprowadzony do pamięci M1, powinno nastąpić wiele odniesień do tego bloku, co powoduje przyspieszenie obsługi.

Aby określić średni czas dostępu do obiektu, musimy rozważyć nie tylko szybkości obu poziomów pamięci, ale także prawdopodobieństwo, że poszukiwane dane mogą być znalezione w pamięci M1. Prawdopodobieństwo to jest znane jako współczynnik trafień. Mamy:

$$T_s = H \times T_1 + (1 - H) \times (T_1 + T_2) = T_1 + (1 - H) \times T_2 \tag{4.1}$$

gdzie:

T_s – średni (systemowy) czas dostępu,

T_1 – czas dostępu do pamięci M1 (np. pamięci podręcznej, w tym dyskowej),

T_2 – czas dostępu do pamięci M2 (np. pamięci głównej, pamięci dyskowej),

H – współczynnik trafień (ułamek czasu, w którym odniesienia są odnajdywane w pamięci M1).

Na rysunku 4.2 jest pokazany średni czas dostępu jako funkcja współczynnika trafień. Jak można zauważyć, przy wysokim procencie trafień średni całkowity czas dostępu jest o wiele bliższy czasowi dostępu do pamięci M1 niż do M2.

Wydajność

Popatrzmy na pewne parametry istotne przy ocenie funkcjonowania pamięci dwupoziomowej. W pierwszej kolejności rozważmy koszt. Mamy:

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \quad (4.2)$$

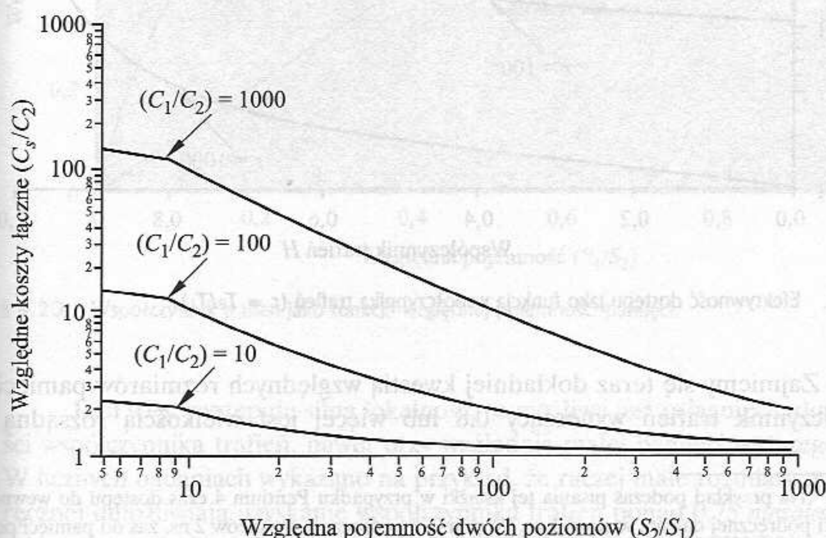
gdzie:
C_s – średni koszt na bit pamięci dwupoziomowej,
C_1 – średni koszt na bit pamięci wyższego poziomu – M1,
C_2 – średni koszt na bit pamięci niższego poziomu – M2,
S_1 – rozmiar pamięci M1,
S_2 – rozmiar pamięci M2.

Chcielibyśmy, aby $C_s = C_2$. Ponieważ $C_1 \gg C_2$, wymagane jest $S_1 \ll S_2$. Zależność tę widać na rys. 4.18.

Rozważmy następnie czas dostępu. Żeby osiągnąć znaczącą poprawę wydajności przez wprowadzenie pamięci dwupoziomowej, trzeba osiągnąć T_s równe w przybliżeniu T_1 ($T_s \approx T_1$). Ponieważ T_1 jest o wiele mniejszy niż T_2 ($T_1 \ll T_2$), wymagany jest współczynnik trafień bliski 1.

Tak więc chcielibyśmy, żeby pamięć M1 była mała w celu ograniczenia kosztów, a jednocześnie żeby była duża w celu poprawienia współczynnika trafień i przez to wydajności. Czy istnieje rozmiar pamięci M1 satysfakcjonujący oba wymagania w rozsądnym stopniu? Możemy odpowiedzieć na to pytanie, posługując się szeregiem pytań pomocniczych:

- Jaka wartość współczynnika trafień jest wymagana, aby $T_s \approx T_1$?
- Jaki rozmiar pamięci M1 zapewni wymagany współczynnik trafień?
- Czy przy tym rozmiarze będzie utrzymany dopuszczalny poziom kosztów?

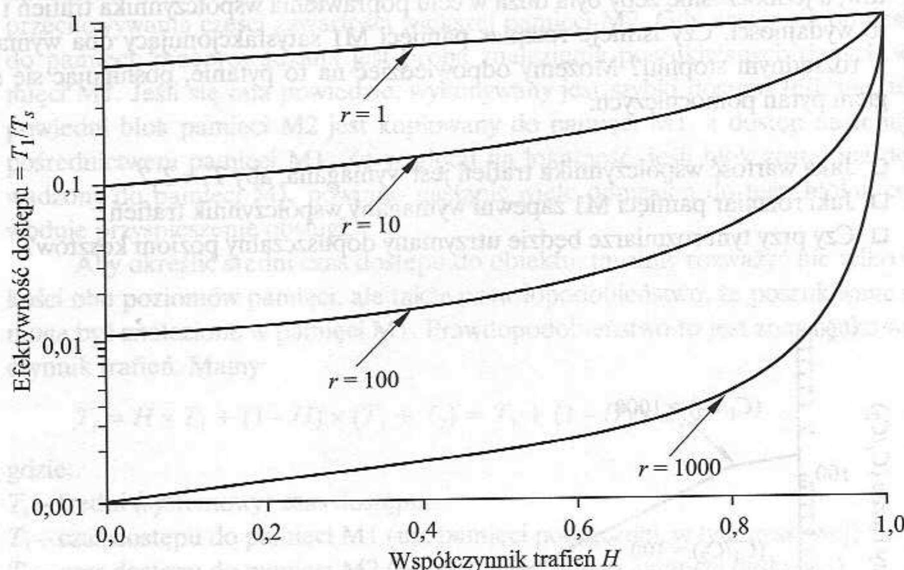


Rysunek 4.18. Zależność między przeciętnym kosztem pamięci a względną pojemnością w przypadku pamięci dwupoziomowej

Żeby na nie odpowiedzieć, rozważmy wielkość T_1/T_s, określaną jako *efektywność dostępu*. Jest ona miarą tego, jak bliski jest średni czas dostępu (T_s) czasowi dostępu do pamięci M1 (T_1). Na podstawie równania (4.1) możemy napisać:

$$\frac{T_1}{T_2} = \frac{1}{1+(1-H)\times\left(\frac{T_2}{T_1}\right)} \tag{4.3}$$

Na rysunku 4.19 widać wykres T_1/T_s w funkcji współczynnika trafień H, przy czym parametrem jest wielkość T_2/T_1. Zwykle czas dostępu do wewnętrznej pamięci podręcznej jest około 25 do 50 razy krótszy od czasu dostępu do pamięci głównej (tzn. T_2/T_1 wynosi 5 do 10), czas dostępu do zewnętrznej pamięci podręcznej jest 5 do 15 razy krótszy od czasu dostępu do pamięci głównej (tzn. T_2/T_1 = 5 do 15)[6], a czas dostępu do pamięci głównej jest około 1000 razy krótszy niż czas dostępu do pamięci dyskowej (T_2/T_1 = 1000). Tak więc, wymagany wzrost wydajności będzie osiągnięty, jeśli współczynnik trafień będzie bliski 0,9.



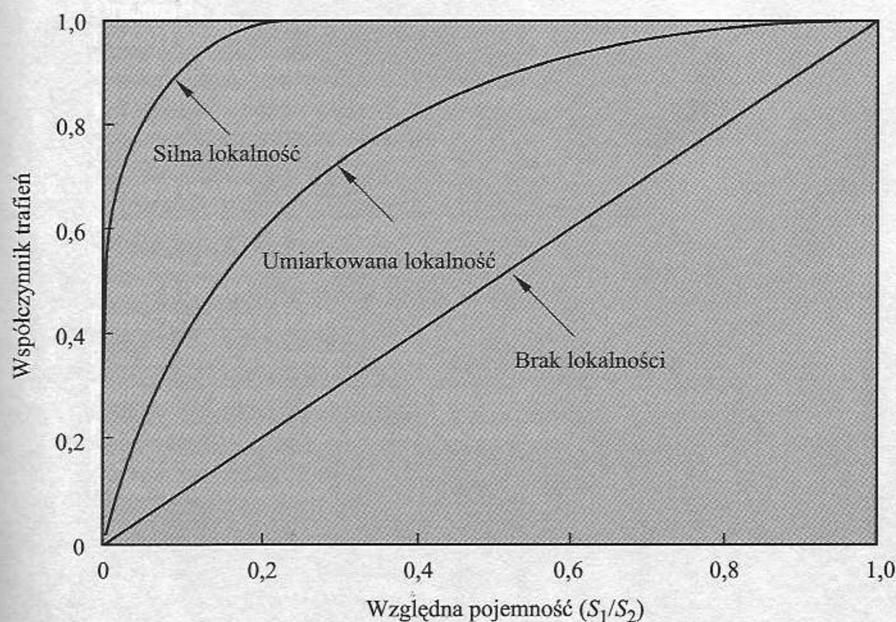
Rysunek 4.19. Efektywność dostępu jako funkcja współczynnika trafień ($r = T_2/T_1$)

Zajmiemy się teraz dokładniej kwestią względnych rozmiarów pamięci. Czy współczynnik trafień wynoszący 0,8 lub więcej jest wielkością rozsądną przy

---

[6] Na przykład podczas pisania tej książki w przypadku Pentium 4 czas dostępu do wewnętrznej pamięci podręcznej danych wynosił 1 ns, do pamięci podręcznej rozkazów 2 ns, zaś do pamięci podręcznej L2 3,5 ns; czas dostępu do pamięci głównej wynosił 30 ns. W przypadku Itanium czas dostępu do wewnętrznej pamięci podręcznej L1 wynosił 2 ns, zaś do pamięci L2 6 ns; do znajdującej się poza mikroukładem pamięci podręcznej L3 wynosił 21 ns; czas dostępu do pamięci głównej – 50 ns.

$S1 \ll S2$? Będzie to zależało od wielu czynników, włącznie z naturą wykorzystywanego oprogramowania oraz ze szczegółami projektowymi pamięci dwupoziomowej. Główną determinantą jest oczywiście stopień lokalności. Rysunek 4.20 ilustruje wpływ lokalności na współczynnik trafień. Oczywiście, jeżeli pamięć M1 ma taki sam rozmiar jak M2, to współczynnik trafień wyniesie 1,0: cała zawartość pamięci M2 jest zawsze przechowywana również w pamięci M1. Załóżmy teraz, że lokalność nie występuje; oznacza to, że odniesienia są całkowicie przypadkowe. Wówczas współczynnik trafień powinien być ściśle liniową funkcją względnego rozmiaru pamięci. Jeśli na przykład pamięć M1 jest o połowę mniejsza niż M2, to w dowolnej chwili połowa zawartości pamięci M2 znajduje się również w M1, a współczynnik trafień wyniesie 0,5. W praktyce jednak odniesienia wykazują pewien stopień lokalności. Wpływ umiarkowanej i silnej lokalności jest pokazany na rysunku.



Rysunek 4.20. Współczynnik trafień jako funkcja względnej pojemności pamięci

Jeśli więc występuje silna lokalność, to możliwe jest osiągnięcie dużych wartości współczynnika trafień, nawet przy względnie małej pamięci wyższego poziomu. W licznych badaniach wykazano na przykład, że raczej małe rozmiary pamięci podręcznej umożliwiają uzyskanie współczynnika trafień ponad 0,75 *niezależnie od rozmiaru pamięci głównej* (patrz np. [AGAR89], [PRZY88], [STRE83] i [SMIT82]). Pamięć podręczna o pojemności od 1 K do 128 K słów jest na ogół wystarczająca, natomiast pojemność pamięci głównej wynosi zwykle wiele MB. Gdy będziemy roz-

patrywać pamięć wirtualną i dyskową pamięć podręczną, zacytujemy inne badania potwierdzające to samo zjawisko: względnie mała pamięć M1 z powodu lokalności umożliwia uzyskanie dużego współczynnika trafień.

Dochodzimy w ten sposób do ostatniego z wcześniej postawionych pytań: czy względny rozmiar obu pamięci pozwala na utrzymanie dopuszczalnych kosztów? Odpowiedź jest oczywiście twierdząca. Jeśli potrzebujemy tylko względnie małej pamięci wyższego poziomu, aby osiągnąć dobrą wydajność, to średni koszt na bit obu poziomów pamięci zbliży się do tańszej pamięci niższego poziomu.

Proszę o zwrócenie uwagi, że przy uwzględnieniu pamięci L2, a tym bardziej L2 i L3, analiza staje się znacznie bardziej złożona. Zagadnienie to zostało przedstawione w [PEIR99] i [HAND98].

Rozdział 5

Pamięć wewnętrzna

5.1. Półprzewodnikowa pamięć główna
Organizacja
DRAM i SRAM
Rodzaje pamięci stałych
Układy logiczne mikroukładów pamięciowych
Obudowy mikroukładów pamięciowych
Organizacja modułów pamięciowych
5.2. Korekcja błędów
5.3. Nowe rozwiązania organizacji DRAM
Synchroniczna pamięć DRAM
Pamięć RDRAM
Pamięć podręczna DRAM
5.4. Polecana literatura i witryny WWW
5.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

PODSTAWOWE SPOSTRZEŻENIA

- Dwiema podstawowymi postaciami pamięci półprzewodnikowych o dostępie swobodnym są: dynamiczna RAM (DRAM) i statyczna RAM (SRAM). Pamięć SRAM jest szybsza, droższa i mniej gęsto upakowana niż DRAM; jest stosowana w pamięciach podręcznych. DRAM jest używana w pamięciach głównych.
- W systemach pamięciowych powszechnie są używane techniki korekcji błędów. Obejmują one nadmiarowe bity będące funkcją bitów danych i stanowiące kod korekcji błędów. Jeśli wystąpi błędny bit, kod ten wykryje go i zwykle naprawi.
- W celu skompensowania stosunkowo małej szybkości DRAM powstało kilka nowych rozwiązań organizacyjnych DRAM. Najszerzej stosowane są: synchroniczna DRAM i RamBus DRAM. W obydwu korzysta się z zegara systemowego, aby zapewnić transfer bloków danych.

Rozdział ten rozpoczynamy od przeglądu półprzewodnikowych podsystemów pamięci głównej, w tym ROM, DRAM i SRAM. Następnie przedstawimy techniki korekcji błędów służące do zwiększania niezawodności pamięci. Na zakończenie poznamy bardziej zaawansowane architektury DRAM.

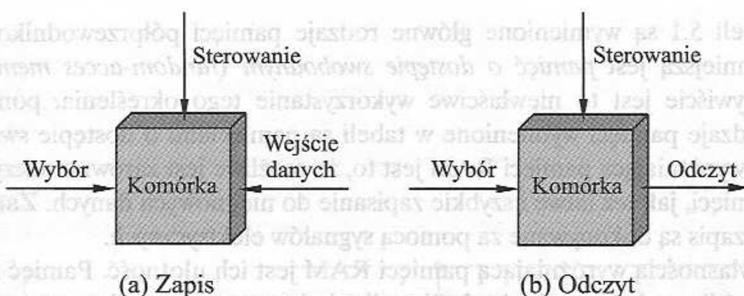
5.1. Półprzewodnikowa pamięć główna

We wcześniejszych komputerach najpowszechniejszą formą komputerowych pamięci głównych o dostępie swobodnym były zespoły pierścieni ferromagnetycznych nazywanych *rdzeniami*. Dlatego też pamięci główne tego rodzaju były nazywane *rdzeniowymi*. Termin ten funkcjonuje do dziś. Dość długo trwało, zanim nadejście mikroelektroniki i jej zalety spowodowały ustąpienie rdzeniowych pamięci magnetycznych. Dzisiaj zastosowanie mikroukładów półprzewodnikowych w pamięciach głównych jest prawie uniwersalne. Głównymi aspektami tej technologii zajmiemy się w następnym punkcie.

Organizacja

Podstawowym elementem pamięci półprzewodnikowej jest komórka pamięci. Chociaż są wykorzystywane różne technologie, wszystkie komórki pamięci półprzewodnikowych mają pewne wspólne cechy:

- Mają dwa stabilne (lub półstabilne) stany, które mogą być użyte do reprezentowania binarnych 1 i 0.
- Umożliwiają zapis (przynajmniej jednokrotny).
- Umożliwiają odczyt.



Rysunek 5.1. Działanie komórki pamięci

Na rysunku 5.1 pokazano działanie komórki pamięci. Najczęściej komórka ma trzy końcówki funkcyjne służące do przenoszenia sygnału elektrycznego. Końcówka wyboru, jak sama nazwa wskazuje, służy do wybierania komórki pamięci w celu przeprowadzenia operacji odczytu lub zapisu. Końcówka sterowania umożliwia wskazanie rodzaju operacji (zapis lub odczyt). W przypadku zapisu przez inną końcówkę wprowadzany jest sygnał elektryczny, który ustala stan komórki na 1 lub 0. W przypadku odczytu ta sama końcówka służy do wyprowadzania sygnału o stanie komórki. Szczegóły organizacji wewnętrznej, funkcjonowania i przebiegów czasowych dotyczące komórki pamięciowej zależą od wykorzystanej technologii układów scalonych i wykraczają poza zakres tej książki. Dla naszych celów przyjmiemy jako dane to, że indywidualne komórki mogą być wybierane w celu przeprowadzenia operacji odczytu i zapisu.

DRAM i SRAM

Wszystkie rodzaje pamięci rozpatrywane w tym punkcie charakteryzują się dostępem swobodnym. Znaczy to, że pojedyncze słowa w pamięci są dostępne bezpośrednio za pomocą wbudowanych układów logicznych adresowania.

Tabela 5.1. Rodzaje pamięci półprzewodnikowych

Rodzaj pamięci	Kategoria	Wymazywanie	Sposób zapisu	Ulotność
Pamięć o dostępie swobodnym (RAM)	odczyt-zapis	elektryczne, na poziomie bajta	elektryczny	ulotna
Pamięć stała (ROM)	tylko odczyt	niemożliwe	maski	nieulotne
Programowalna pamięć stała (PROM)				
Wymazywalna PROM (EPROM)		światłem UV, na poziomie mikroukładu	elektryczny	
Pamięć błyskawiczna	głównie odczyt	elektryczne, na poziomie bloku		
Elektrycznie wymazywalna PROM (EEPROM)		elektryczne, na poziomie bajta		

W tabeli 5.1 są wymienione główne rodzaje pamięci półprzewodnikowych. Najpowszechniejszą jest *pamięć o dostępie swobodnym (random-acces memory* – RAM). Oczywiście jest to niewłaściwe wykorzystanie tego określenia, ponieważ wszystkie rodzaje pamięci wymienione w tabeli są pamięciami o dostępie swobodnym. Cechą wyróżniającą pamięci RAM jest to, że możliwe jest zarówno odczytanie danych z pamięci, jak też łatwe i szybkie zapisanie do niej nowych danych. Zarówno odczyt, jak i zapis są dokonywane za pomocą sygnałów elektrycznych.

Inną własnością wyróżniającą pamięci RAM jest ich ulotność. Pamięć RAM musi mieć źródło stałego zasilania. Jeśli zasilanie jest przerwane, dane są tracone. Pamięć RAM może więc być używana tylko do przechowywania tymczasowego. Dwiema tradycyjnymi postaciami RAM używanymi w komputerach są DRAM i SRAM.

Dynamiczna pamięć RAM

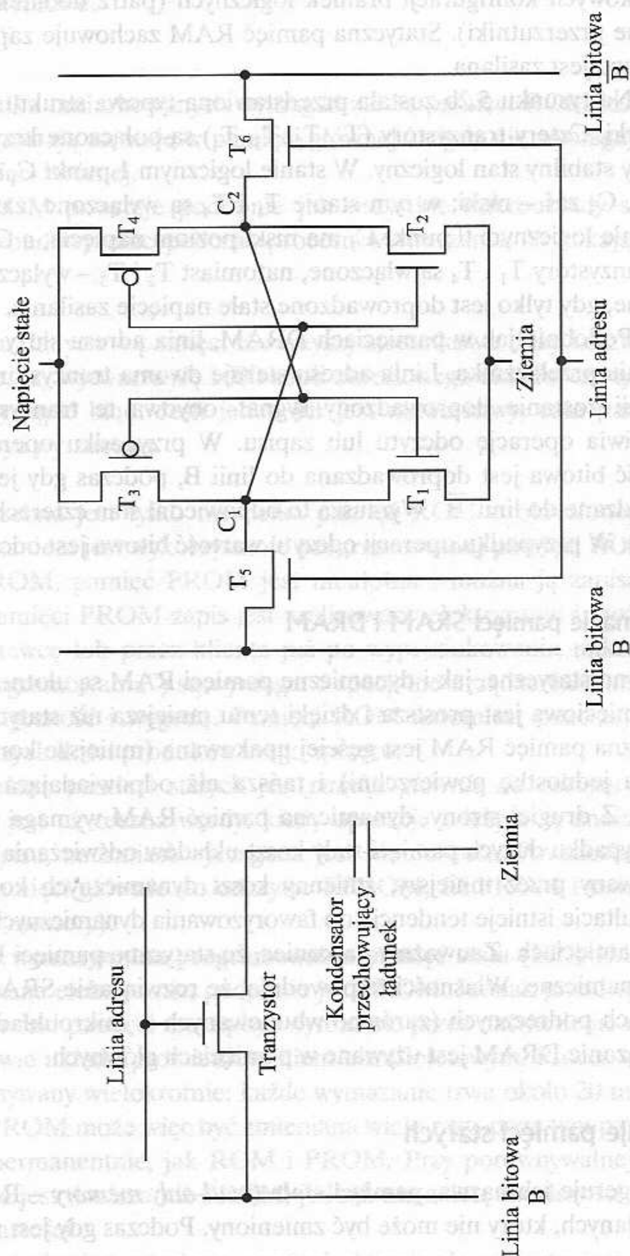
Pamięci RAM można podzielić na statyczne i dynamiczne. Dynamiczna pamięć RAM (DRAM) jest wykonana z komórek, które przechowują dane, podobnie jak kondensatory przechowują ładunek elektryczny. Obecność lub brak ładunku w kondensatorze są interpretowane jako binarne 1 lub 0. Ponieważ kondensatory mają naturalną tendencję do rozładowywania się, dynamiczne pamięci RAM wymagają okresowego odświeżania ładunku w celu zachowania danych. Określenie *dynamiczna* odnosi się do tej właśnie skłonności do zanikania wprowadzonego ładunku, nawet jeśli wciąż jest doprowadzane zasilanie.

Na rysunku 5.2a przedstawiono typową strukturę DRAM pojedynczej komórki służącej do zapisywania jednego bitu. Gdy wartość bitu jest odczytana z tej komórki lub zapisana w niej, wzbudzana jest linia adresu. Tranzystor działa jako przełącznik, który jest zamknięty (i umożliwia przepływ prądu), gdy napięcie jest doprowadzone do linii adresu, oraz otwarty (uniemożliwiając przepływ prądu), jeśli na linii adresu nie ma napięcia.

W przypadku operacji zapisu do linii bitowej jest doprowadzany sygnał napięciowy; wysokie napięcie reprezentuje 1, niskie zaś – 0. Następnie doprowadzany jest sygnał do linii adresu, dzięki czemu ładunek może być przeniesiony do kondensatora.

W przypadku operacji zapisu, gdy zostanie wybrana linia adresu, tranzystor zostaje włączony i ładunek znajdujący się w kondensatorze jest doprowadzany do linii bitowej oraz do wzmacniacza odczytu. Wzmacniacz ten porównuje napięcie kondensatora z wartością odniesienia i na tej podstawie określa, czy dana komórka zawiera logiczną 1, czy też 0. Odczyt zawartości komórki powoduje rozładowanie kondensatora, więc w celu zakończenia operacji odczytu jego stan musi być przywrócony.

Chociaż komórka DRAM służy do zapisywania pojedynczego bitu (0 lub 1), jest w zasadzie przyrządem analogowym. Kondensator może przechowywać dowolny ładunek w pewnym zakresie; na podstawie ustalonej wartości progowej można dopiero określić, czy ładunek zostanie zinterpretowany jako 1, czy jako 0.



Rysunek 5.2. Typowe struktury komórek pamięci

Statyczna pamięć RAM

W przeciwieństwie do tego, statyczna pamięć RAM (SRAM) jest przyrządem cyfrowym, w którym są stosowane takie same elementy logiczne jak w procesorze. W pamięci SRAM wartości binarne są zapisywane przy użyciu tradycyjnych, przerzutnikowych konfiguracji bramek logicznych (patrz dodatek A, w którym zostały opisane przerzutniki). Statyczna pamięć RAM zachowuje zapisane dane tak długo, jak długo jest zasilana.

Na rysunku 5.2b została przedstawiona typowa struktura SRAM pojedynczej komórki. Cztery tranzystory (T_1, T_2, T_3, T_4) są połączone krzyżowo tak, aby zachowywały stabilny stan logiczny. W stanie logicznym 1 punkt C_1 ma wysoki poziom napięcia, C_2 zaś – niski; w tym stanie T_1 i T_4 są wyłączone, zaś T_2 i T_3 – włączone[1]. W stanie logicznym 0 punkt C_1 ma niski poziom napięcia, a C_2 – wysoki; w tym stanie tranzystory T_1 i T_4 są włączone, natomiast T_2 i T_3 – wyłączone. Obydwa stany są stabilne, gdy tylko jest doprowadzone stałe napięcie zasilania.

Podobnie jak w pamięciach DRAM, linia adresu służy do otwierania lub zamykania przełącznika. Linia adresu steruje dwoma tranzystorami (T_5 i T_6). Gdy do tej linii zostanie doprowadzony sygnał, obydwa te tranzystory są włączane, co umożliwia operację odczytu lub zapisu. W przypadku operacji zapisu, pożądana wartość bitowa jest doprowadzana do linii B, podczas gdy jej dopełnienie jest doprowadzane do linii $\overline{B}$. Wymusza to odpowiedni stan czterech tranzystorów (T_1, T_2, T_3, T_4). W przypadku operacji odczytu wartość bitowa jest odczytywana z linii B.

Porównanie pamięci SRAM i DRAM

Zarówno statyczne, jak i dynamiczne pamięci RAM są ulotne. Dynamiczna komórka pamięciowa jest prostsza i dzięki temu mniejsza niż statyczna. W rezultacie dynamiczna pamięć RAM jest gęściej upakowana (mniejsze komórki = więcej komórek na jednostkę powierzchni) i tańsza niż odpowiadająca jej statyczna pamięć RAM. Z drugiej strony, dynamiczna pamięć RAM wymaga układów odświeżania. W przypadku dużych pamięci stały koszt układów odświeżania jest z nawiązką skompensowany przez mniejszy, zmienny koszt dynamicznych komórek pamięciowych. W rezultacie istnieje tendencja do faworyzowania dynamicznych pamięci RAM w dużych pamięciach. Zauważmy na koniec, że statyczne pamięci RAM są nieco szybsze niż dynamiczne. Własności te powodują, że rozwiązanie SRAM jest używane w pamięciach podręcznych (zarówno wbudowanych w mikroukład, jak i poza nim), zaś rozwiązanie DRAM jest używane w pamięciach głównych.

Rodzaje pamięci stałych

Jak sugeruje ich nazwa, **pamięci stałe** (*read-only memory* – ROM) zawierają trwały wzór danych, który nie może być zmieniony. Podczas gdy jest możliwe odczytywanie

[1] Kółka przy bramkach tranzystorów T_3 i T_4 oznaczają negowanie sygnału.

pamięci ROM, nie można zapisać w nich nowych danych. Ważnym zastosowaniem pamięci ROM jest mikroprogramowanie, którym zajmiemy się w części IV. Do innych potencjalnych zastosowań należą:

- podprogramy biblioteczne dla często używanych funkcji,
- programy systemowe,
- tablice funkcji.

W przypadku umiarkowanych wymagań zaletą pamięci ROM jest to, że dane lub program pozostają na stałe w pamięci głównej i nigdy nie wymagają ładowania z urządzeń pamięci wtórnej.

Pamięć ROM powstaje podobnie jak wszystkie mikroukłady scalone, przy czym dane są wbudowywane podczas procesu wytwarzania. Wynikają z tego dwa problemy.

- ❑ Umieszczaniu danych w pamięci towarzyszy stosunkowo duży koszt stały, niezależny od tego, czy wytwarza się 1000 kopii określonego układu, czy tylko jedną.
- ❑ Nie może wystąpić błąd. Jeśli jeden bit jest niewłaściwy, cała partia układów ROM musi być wyrzucona.

Jeśli potrzebna jest tylko niewielka pamięć ROM o określonej zawartości, tańszym rozwiązaniem jest wykorzystanie **programowalnej pamięci ROM** (PROM). Podobnie jak ROM, pamięć PROM jest nieulotna i można ją zapisać tylko raz. W przypadku pamięci PROM zapis jest realizowany elektrycznie i może być wykonany przez dostawcę lub przez klienta już po wyprodukowaniu mikroukładu. Do zapisu (lub „programowania") są wymagane specjalne urządzenia. Pamięci PROM zapewniają elastyczność i wygodę. Pamięci ROM natomiast pozostają atrakcyjne w przypadku dużych ilości produkowanego sprzętu.

Inną odmianą pamięci stałych jest *pamięć głównie do odczytu* (*read-mostly memory*), która jest użyteczna wtedy, kiedy operacje odczytu są znacznie częstsze niż operacje zapisu, natomiast wymagana jest nieulotność. Powszechnie znane są trzy odmiany pamięci głównie do odczytu: EPROM, EEPROM i tzw. *pamięć błyskawiczna* (*flash memory*).

Optycznie **wymazywalna programowalna pamięć stała** (EPROM) jest odczytywana i zapisywana elektrycznie, podobnie jak PROM. Jednak przed operacją zapisu wszystkie komórki pamięci muszą być wymazane przez naświetlenie znajdującego się już w obudowie układu promieniowaniem ultrafioletowym. Proces wymazywania może być wykonywany wielokrotnie; każde wymazanie trwa około 20 minut. Zawartość pamięci EPROM może więc być zmieniana wiele razy, poza tym pamięć ta przechowuje dane permanentnie, jak ROM i PROM. Przy porównywalnej pojemności pamięć EPROM jest droższa niż PROM, jednak ma zaletę możliwości wielokrotnej aktualizacji zawartości.

Bardziej atrakcyjną odmianą pamięci głównie do odczytu jest **elektrycznie wymazywalna programowalna pamięć stała** (EEPROM). Jest to pamięć głównie

do odczytu, która może być zapisana bez wymazywania poprzedniej zawartości; aktualizowany jest tylko bajt (lub bajty) adresowane. Operacja zapisu trwa znacznie dłużej niż odczytu i zajmuje czas rzędu kilkuset mikrosekund na bajt. EEPROM łączy zaletę nieulotności z możliwością aktualizacji na miejscu, przy wykorzystaniu zwykłych magistralowych linii sterowania, adresów i danych. EEPROM jest droższa niż EPROM, a także mniej gęsto upakowana (zawiera mniej bitów w mikroukładzie).

Kolejną postacią pamięci półprzewodnikowej jest **pamięć błyskawiczna** (nazwana tak z powodu szybkości, z jaką może być reprogramowana). Wprowadzono ją po raz pierwszy w połowie lat osiemdziesiątych. Zajmuje miejsce pośrednie między EPROM i EEPROM zarówno pod względem kosztu, jak i funkcjonalności. Podobnie jak w EEPROM w pamięci błyskawicznej wykorzystuje się metodę wymazywania elektrycznego. Cała pamięć błyskawiczna może być wymazana w ciągu kilku sekund, a więc o wiele szybciej niż EPROM. Ponadto możliwe jest wymazywanie bloków pamięci zamiast całego mikroukładu. Jednak pamięć błyskawiczna nie umożliwia wymazywania na poziomie bajtów. Podobnie jak w EPROM w pamięci błyskawicznej wykorzystuje się tylko jeden tranzystor na bit, osiągając przez to (w porównaniu do EEPROM) wysoką gęstość upakowania, podobną do EPROM.

Układy logiczne mikroukładów pamięciowych

Podobnie jak inne układy scalone, pamięć półprzewodnikowa ma postać obudowanego mikroukładu (rys. 2.7). Każdy mikroukład zawiera zestaw komórek pamięci.

Jak widzieliśmy, w hierarchicznym układzie pamięci występują współzależności między szybkością, pojemnością a kosztem. Takie współzależności istnieją również, jeśli rozpatrujemy organizację komórek pamięci i funkcjonalne układy logiczne zawarte w mikroukładzie. W przypadku pamięci półprzewodnikowych jednym z kluczowych problemów projektowych jest liczba bitów danych, które mogą być jednocześnie odczytywane lub zapisywane. Na jednym biegunie znajduje się organizacja, w której fizyczne uporządkowanie komórek w układzie jest takie, jak uporządkowanie logiczne (postrzegane przez procesor) słów w pamięci. Komórki są wtedy zorganizowane w postaci *W* słów *B*-bitowych. Na przykład mikroukład 16 Mbit może być zorganizowany jako 1 M (milion) słów 16-bitowych. Na drugim biegunie znajduje się organizacja 1-bitowa, w przypadku której w określonym czasie odczytuje się lub zapisuje tylko jeden bit. Przedstawimy organizację mikroukładów pamięciowych na przykładzie pamięci DRAM; organizacja pamięci ROM jest podobna, choć prostsza.

Na rysunku 5.3 jest pokazana typowa organizacja 16-megabitowej pamięci DRAM. W tym przypadku jednocześnie mogą być odczytywane lub zapisywane 4 bity. Logicznie rzecz biorąc, zespół pamięci jest zorganizowany w postaci 4 kwadratowych układów 2048 na 2048 elementów. Możliwe są różne organizacje fizyczne. W każdym przypadku elementy zespołu są połączone zarówno przez linie poziome (wiersze), jak i pionowe (kolumny). Każda linia pozioma jest połączona

z końcówkami „wybór" każdej komórki w wierszu; każda linia pionowa jest połączona z końcówkami „zapis/odczyt" każdej komórki w kolumnie.

Linie adresowe dostarczają adres słowa, które ma być wybrane. Łącznie potrzeba $\log_2 W$ linii. W naszym przykładzie wymaganych jest 11 linii w celu wybrania 1 z 2048 wierszy. Linie te są doprowadzone do dekodera wiersza mającego 11 linii wejściowych i 2048 wyjściowych. Układ logiczny dekodera aktywuje jedną z 2048 linii zależnie od wzoru bitowego na 11 liniach wejściowych ($2^{11} = 2048$).

Dodatkowe 11 linii adresu wybiera 1 z 2048 kolumn, przy czym na 1 kolumnę przypadają 4 bity. W celu wprowadzania i wyprowadzania 4 bitów z bufora danych są wykorzystywane 4 linie danych. Na wejściu (zapis) sterownik bitowy każdej linii bitu jest aktywowany do 1 lub 0, zależnie od wartości odpowiedniej linii danych. Na wyjściu (odczyt) sygnał z każdej linii bitowej jest doprowadzany poprzez wzmacniacz odczytu do linii danych. Za pomocą linii wiersza dokonuje się wyboru, który wiersz komórek jest wykorzystywany przy odczycie lub zapisie.

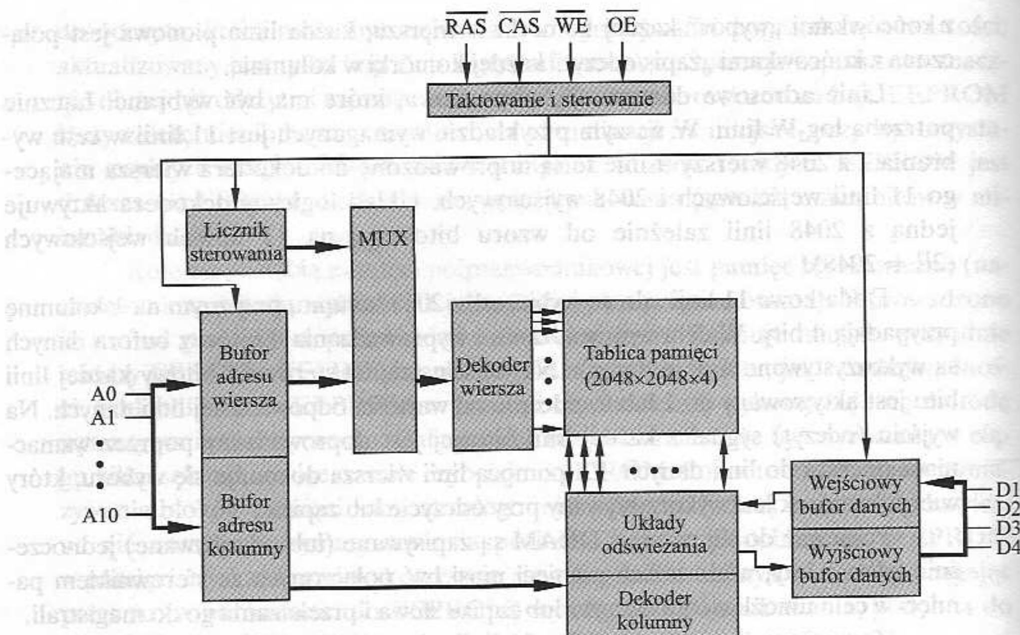
Ponieważ do tej pamięci DRAM są zapisywane (lub odczytywane) jednocześnie tylko 4 bity, wiele takich pamięci musi być połączonych ze sterownikiem pamięci w celu umożliwienia odczytu lub zapisu słowa i przekazania go do magistrali.

Zauważmy, że występuje tylko 11 linii adresu (A0÷A10), połowa liczby, której moglibyśmy się spodziewać dla zespołu 2048×2048. Czyni się tak, aby zmniejszyć liczbę doprowadzeń do układu. Wymagane linie adresu (22) przechodzą przez układy logiczne wyboru zewnętrzne w stosunku do mikroukładu, po czym są multipleksowane do 11 linii adresu. Najpierw do mikroukładu doprowadza się 11 sygnałów adresowych określających adres wiersza w zespole, po czym 11 pozostałych sygnałów adresowych określa adres kolumny. Sygnałom tym towarzyszą sygnały: wyboru adresu wiersza ($\overline{RAS}$) i wyboru adresu kolumny ($\overline{CAS}$) umożliwiające sterowanie czasowe mikroukładu.

Końcówki zezwolenia zapisu ($\overline{WE}$) i zezwolenia wyjścia ($\overline{OE}$) określają, czy jest realizowana operacja zapisu, czy odczytu. Dwie inne końcówki – nie pokazane na rys. 5.3 – to ziemia (V_{SS}) i zasilanie (V_{CC}).

Jako efekt uboczny, adresowanie multipleksowane w połączeniu z kwadratową organizacją zespołów komórek umożliwiło 4-krotne zwiększenie pojemności pamięci w każdej nowej generacji mikroukładów pamięciowych. Jedno dodatkowe doprowadzenie przeznaczone do adresowania podwaja liczbę wierszy i kolumn, przez co rozmiar mikroukładu pamięciowego wzrasta 4-krotnie.

Na rysunku 5.3 są również pokazane układy odświeżania. Wszystkie pamięci DRAM wymagają operacji odświeżania. Prostym sposobem odświeżania jest uniemożliwienie używania mikroukładu DRAM w czasie, gdy wszystkie komórki danych są odświeżane. Licznik odświeżania przechodzi kolejno przez wszystkie wiersze. W przypadku każdego wiersza linie wyjściowe licznika odświeżania są łączone z dekoderem wiersza i wzbudzana jest linia RAS. Powoduje to odświeżenie każdej komórki w wierszu.



Rysunek 5.3. Typowa pamięć DRAM o pojemności 16 Mbit (4 M × 4)

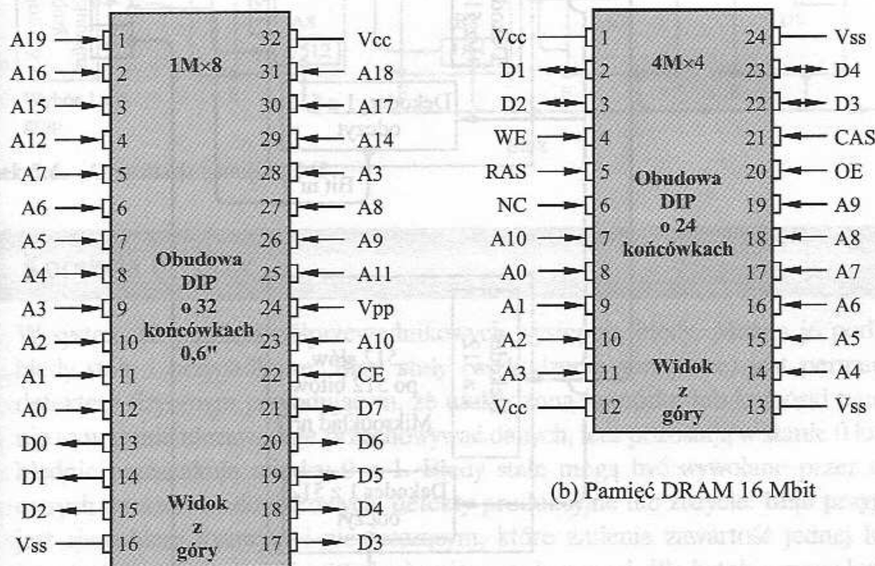
Obudowy mikroukładów pamięciowych

Jak stwierdziliśmy w rozdz. 2, układ scalony jest montowany w obudowie, która zawiera końcówki służące do łączenia ze światem zewnętrznym.

Na rysunku 5.4a jest pokazana przykładowa obudowa pamięci EPROM o pojemności 8 Mbit, zorganizowanej jako 1 M × 8. W tym przypadku organizacja ta może być traktowana jako oparta na słowach. Obudowa ma 32 końcówki, co jest jednym z rozwiązań standardowych. Końcówkom są przypisywane następujące linie sygnałowe:

- adres słowa; w przypadku 1 M słów wymagane jest 20 (2^{20} = 1 M) końcówek (A0÷A19);
- odczytywane dane, wymagające 8 linii (D0÷D7);
- zasilanie mikroukładu (V_{CC});
- uziemienie (V_{SS});
- uaktywnienie mikroukładu (CE); ponieważ może występować wiele mikroukładów pamięciowych, z których każdy jest dołączony do tej samej szyny adresowej, końcówka CE jest wykorzystywana do określania, czy adres dotyczy danego mikroukładu; końcówka CE jest pobudzana przez układy logiczne dołączone do linii najbardziej znaczących bitów szyny adresowej (tzn. do linii bitów powyżej A19);
- napięcie programowania (V_{PP}), które jest doprowadzane podczas programowania (operacje zapisu).

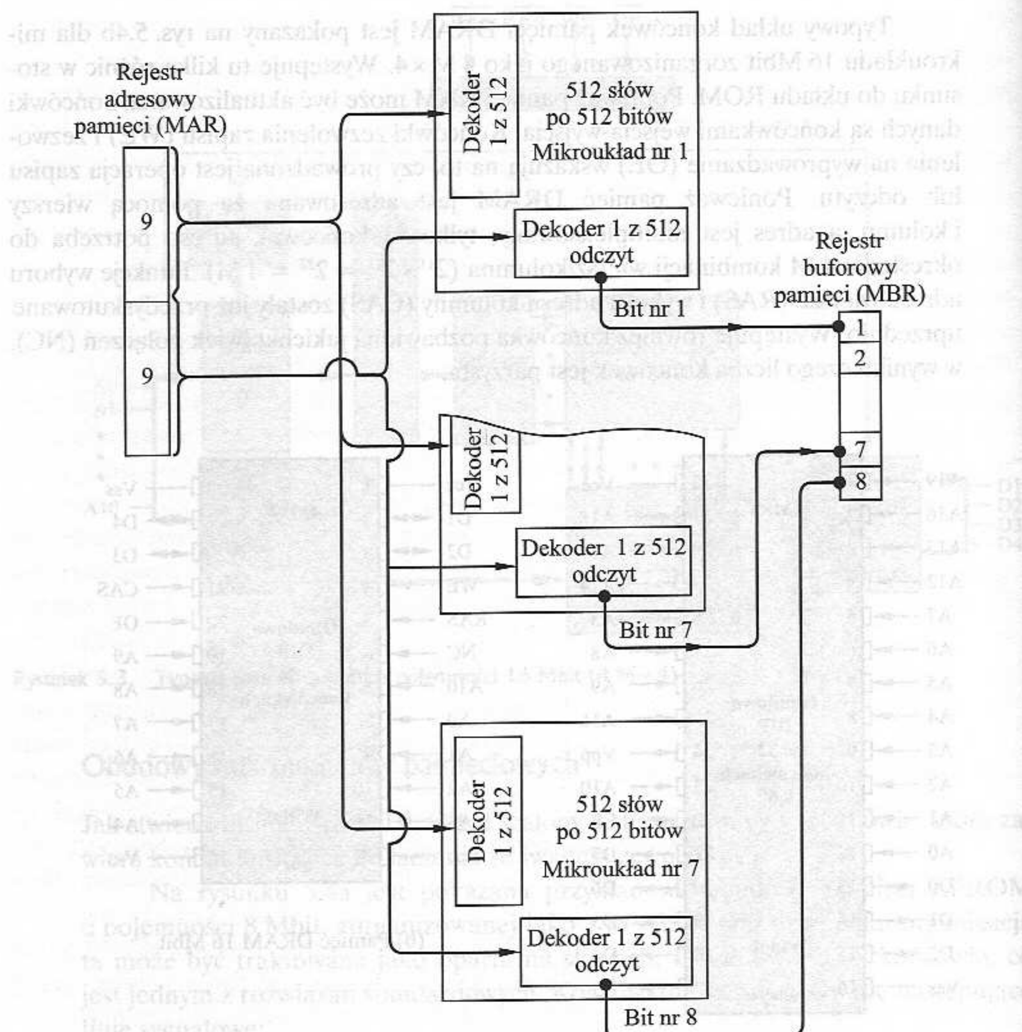
Typowy układ końcówek pamięci DRAM jest pokazany na rys. 5.4b dla mikroukładu 16 Mbit zorganizowanego jako 4 M × 4. Występuje tu kilka różnic w stosunku do układu ROM. Ponieważ pamięć RAM może być aktualizowana, końcówki danych są końcówkami wejścia-wyjścia. Końcówki zezwolenia zapisu (WE) i zezwolenia na wyprowadzanie (OE) wskazują na to, czy prowadzona jest operacja zapisu lub odczytu. Ponieważ pamięć DRAM jest adresowana za pomocą wierszy i kolumn, a adres jest multipleksowany, tylko 11 końcówek adresu potrzeba do określenia 4 M kombinacji wiersz/kolumna ($2^{11} \times 2^{11} = 2^{22} = 4$ M). Funkcje wyboru adresu wiersza (RAS) i wyboru adresu kolumny (CAS) zostały już przedyskutowane uprzednio. Występuje również końcówka pozbawiona jakichkolwiek połączeń (NC), w wyniku czego liczba końcówek jest parzysta.



Rysunek 5.4. Rozkład wyprowadzeń i sygnały typowej obudowy mikroukładu pamięciowego

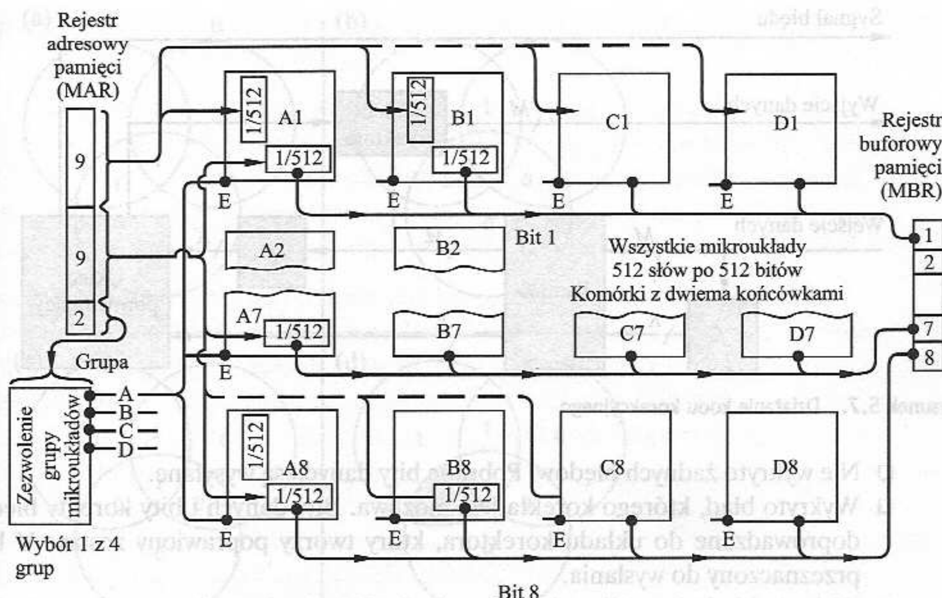
Organizacja modułów pamięciowych

Jeśli mikroukład RAM zawiera tylko 1 bit na słowo, będziemy oczywiście potrzebowali przynajmniej liczby mikroukładów równej liczbie bitów w słowie. Na rysunku 5.5 jest pokazane przykładowo, jak może być zorganizowany moduł pamięci zawierający 256 K słów 8-bitowych. W przypadku 256 K słów wymagany jest adres 18-bitowy; jest on dostarczany do modułu z pewnego źródła zewnętrznego (np. z linii adresowych magistrali, do której moduł jest dołączony). Adres jest doprowadzany do 8 mikroukładów 256 K × 1, z których każdy umożliwia wejście-wyjście 1 bitu.



Rysunek 5.5. Organizacja pamięci 256 KB

Organizacja taka jest odpowiednia, dopóki rozmiar pamięci mierzony w słowach jest równy liczbie bitów w mikroukładzie. W przypadku gdy wymagana jest większa pamięć, potrzebna jest matryca mikroukładów. Na rysunku 5.6 jest pokazana możliwa organizacja pamięci składającej się z 1 M słów 8-bitowych. W tym przypadku występują 4 kolumny mikroukładów; każda kolumna zawiera 256 K słów uporządkowanych w sposób pokazany na rys. 5.5. W przypadku 1 M słów wymaganych jest 20 linii adresowych. Osiemnaście najmniej znaczących bitów doprowadza się do wszystkich 32 modułów. Dwa najbardziej znaczące bity są doprowadzone do modułu logicznego wyboru grupy, który wysyła sygnał uaktywnienia mikroukładu do jednej z 4 kolumn modułów.



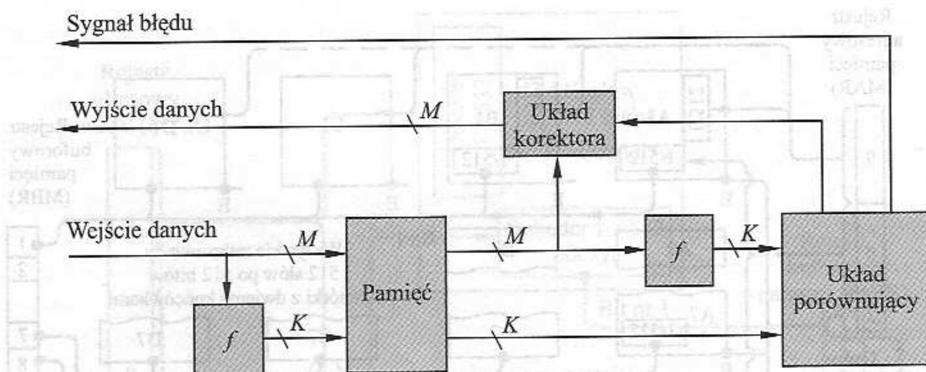
Rysunek 5.6. Organizacja pamięci 1 MB

5.2. Korekcja błędów

W systemach pamięci półprzewodnikowych występują błędy. Można je podzielić na błędy stałe i przypadkowe. **Błąd stały** (uszkodzenie sprzętowe) jest permanentnym defektem fizycznym powodującym, że uszkodzona komórka lub komórki pamięciowe nie są w stanie niezawodnie przechowywać danych, lecz pozostają w stanie 0 lub 1, albo błędnie przeskakują między 0 a 1. Błędy stałe mogą być wywołane przez działanie ostrych narażeń środowiskowych, defekty produkcyjne lub zużycie. **Błąd przypadkowy** jest zjawiskiem losowym i nieniszczącym, które zmienia zawartość jednej lub wielu komórek pamięciowych, bez uszkadzania samej pamięci. Błędy takie mogą być spowodowane przez problemy zasilania lub cząsteczki alfa. Cząsteczki te są rezultatem rozpadu promieniotwórczego i są niebezpiecznie powszechne, ponieważ jądra promieniotwórcze znajdują się w małych ilościach prawie we wszystkich materiałach. Zarówno błędy stałe, jak i przypadkowe są oczywiście niepożądane i większość współczesnych systemów pamięci głównych zawiera układy logiczne wykrywające i korygujące błędy.

Na rysunku 5.7 jest pokazane w sposób ogólny, jak ten proces jest przeprowadzany. Gdy dane mają być wczytane do pamięci, przeprowadza się na tych danych obliczenia, określane jako funkcja f, w celu utworzenia kodu. Zarówno kod, jak i dane są przechowywane. W rezultacie, jeśli ma być zapisane M-bitowe słowo danych, a kod ma długość K bitów, to aktualna długość przechowywanego słowa wynosi $M + K$ bitów.

Gdy uprzednio zmagazynowane słowo jest odczytywane, kod jest wykorzystywany do wykrywania i ewentualnej korekty błędów. Generowany jest nowy zestaw K bitów kodowych z M bitów danych, po czym porównuje się go z pobranymi bitami kodowymi. Porównanie prowadzi do jednego z trzech wyników:



Rysunek 5.7. Działanie kodu korekcyjnego

- Nie wykryto żadnych błędów. Pobrane bity danych są wysyłane.
- Wykryto błąd, którego korekta jest możliwa. Bity danych i bity korekty błędu są doprowadzane do układu korektora, który tworzy poprawiony zestaw M bitów przeznaczony do wysłania.
- Wykryto błąd niemożliwy do poprawienia. Stan ten jest zgłaszany.

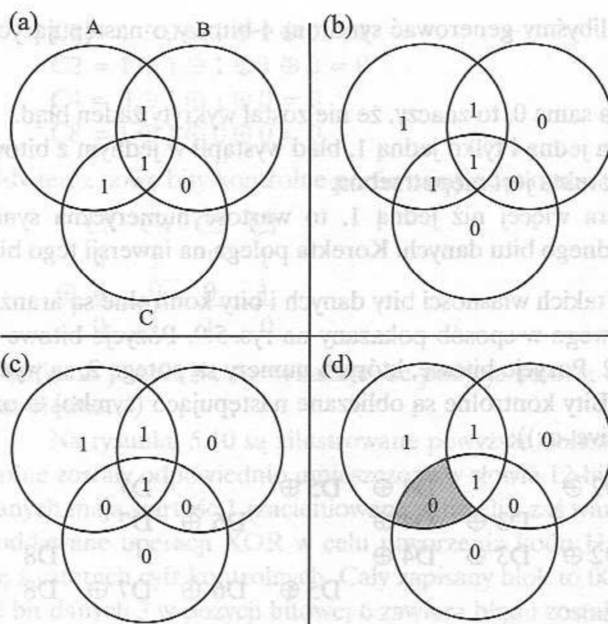
Kody funkcjonujące w ten sposób są określane jako *kody korekcyjne*. Kod jest charakteryzowany przez liczbę błędów bitowych w słowie, którą może on wykryć i poprawić.

Najprostszym kodem korekcyjnym jest *kod Hamminga*, opracowany przez Richarda Hamminga z Bell Laboratories. Na rysunku 5.8 do zilustrowania wykorzystania tego kodu w odniesieniu do słów 4-bitowych ($M = 4$) zostały wykorzystane wykresy Venna. W przypadku trzech krzyżujących się okręgów mamy do czynienia z siedmioma przedziałami. Przypisujemy 4 bity danych przedziałom wewnętrznym (rys. 5.8a). Pozostałe przedziały są wypełniane tzw. *bitami parzystości*. Bit parzystości jest wybierany tak, że całkowita liczba jedynek w jego okręgu jest parzysta (rys. 5.8b). Wobec tego, ponieważ okrąg A zawiera 3 jedynki danych, bit parzystości w tym okręgu jest ustawiany na 1. Jeśli teraz błąd spowoduje zmianę jednego z bitów danych (rys. 5.8c), jest on łatwy do wykrycia. Sprawdzając bity parzystości, wykrywamy sprzeczności w okręgach A i C, ale nie w B. Tylko jeden z 7 przedziałów znajduje się w A i C, ale nie w B. Błąd może więc być poprawiony przez zmianę tego bitu.

W celu wyjaśnienia wykorzystanej koncepcji opracujemy kod, który może posłużyć do wykrycia i skorygowania 1-bitowego błędu w słowach 8-bitowych.

Na wstępie określmy wymaganą długość kodu. Zgodnie z rys. 5.7 logiczne układy porównujące otrzymują na wejściu dwie wartości K-bitowe. Porównanie bit po bicie jest przeprowadzane przy użyciu bramki EXOR (*exclusive-or*, LUB wykluczające) o 2 wejściach. Wynik jest określany jako *słowo-syndrom*. Tak więc, każdy bit syndromu jest 0 lub 1 zależnie od tego, czy jest, czy też nie ma zgodności bitów na dwóch wejściach.

Słowo-syndrom ma więc K bitów i zakres wartości między 0 a $2^K - 1$. Wartość 0 wskazuje, że nie został wykryty żaden błąd, zaś $2^K - 1$ wartości służy do wskazania



Rysunek 5.8. Kod korekcyjny Hamminga

błędu i jego lokalizacji bitowej. Ponieważ błąd może wystąpić w każdym z M bitów danych lub K bitów kontrolnych, musimy mieć

$$2^K - 1 \geqslant M + K$$

Nierówność ta określa liczbę bitów wymaganą do skorygowania błędu 1-bitowego w słowie zawierającym M bitów danych. Na przykład w przypadku 8-bitowego słowa danych ($M = 8$) mamy

- $K = 3$: $2^3 - 1 < 8 + 3$
- $K = 4$: $2^4 - 1 > 8 + 4$

Osiem bitów danych wymaga zatem czterech bitów kontrolnych. W pierwszych trzech kolumnach tabeli 5.2 jest podane, ile liczb bitów kontrolnych jest wymaganych w przypadku słów o różnej długości.

Tabela 5.2. Wzrost długości słowa po uwzględnieniu korekty błędu

Liczba bitów danych	Poprawianie pojedynczego błędu		Poprawianie pojedynczego błędu, wykrywanie podwójnego błędu	
	Bity kontrolne	% wzrostu	Bity kontrolne	% wzrostu
8	4	50	5	62,5
16	5	31,25	6	37,5
32	6	18,75	7	21,875
64	7	10,94	8	12,5
128	8	6,25	9	7,03
256	9	3,52	10	3,91

Dla wygody chcielibyśmy generować syndrom 4-bitowy o następujących własnościach:

- Jeśli syndrom zawiera same 0, to znaczy, że nie został wykryty żaden błąd.
- Jeśli syndrom zawiera jedną i tylko jedną 1, błąd wystąpił w jednym z bitów kontrolnych. Wówczas korekta jest niepotrzebna.
- Jeśli syndrom zawiera więcej niż jedną 1, to wartość numeryczna syndromu wskazuje pozycję błędnego bitu danych. Korekta polega na inwersji tego bitu.

W celu uzyskania takich własności bity danych i bity kontrolne są aranżowane w postaci słowa 12-bitowego w sposób pokazany na rys. 5.9. Pozycje bitowe są ponumerowane od 1 do 12. Pozycje bitowe, których numery są potęgą 2, są wyznaczone jako bity kontrolne. Bity kontrolne są obliczane następująco (symbol ⊕ oznacza operację EXOR (exclusive–or)):

$$
\begin{aligned}
C1 &= D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \\
C2 &= D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \\
C4 &= D2 \oplus D3 \oplus D4 \oplus D8 \\
C8 &= D5 \oplus D6 \oplus D7 \oplus D8
\end{aligned}
$$

Pozycja bitowa	12	11	10	9	8	7	6	5	4	3	2	1
Numer pozycji	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit danych	D8	D7	D6	D5		D4	D3	D2		D1		
Bit kontrolny					C8				C4		C2	C1

Rysunek 5.9. Rozkład bitów danych i bitów kontrolnych

Każdy bit kontrolny działa na każdej pozycji bitu danych, której numer zawiera 1 w odpowiedniej pozycji kolumny. Tak więc pozycje bitowe danych 3, 5, 7, 9 i 11 (D1, D2, D4, D5, D7) zawierają 1 w najmniej znaczącym bicie swojego numeru pozycji, podobnie jak C1; pozycje bitowe 3, 6, 7, 10 i 11 zawierają człon 1 w drugiej pozycji bitowej, podobnie jak C2 i tak dalej. Inaczej mówiąc, pozycja bitowa n jest sprawdzana przez bity C_i takie, że $\sum i = n$. Na przykład pozycja 7 jest sprawdzana przez bity znajdujące się na pozycjach 4, 2 i 1; zachodzi więc 7 = 4 + 2 + 1.

Sprawdźmy na przykładzie, jak funkcjonuje ten schemat. Załóżmy, że 8-bitowym słowem wejściowym jest 00111001, z bitem danych D1 na najbardziej znaczącej pozycji. Obliczenia są następujące:

$$
\begin{aligned}
C1 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
C2 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
C4 &= 0 \oplus 0 \oplus 1 \oplus 0 = 1 \\
C8 &= 1 \oplus 1 \oplus 0 \oplus 0 = 0
\end{aligned}
$$

Załóżmy teraz, że bit danych 3 zawiera błąd i jest zmieniony z 0 na 1. Po ponownym przeliczeniu bitów kontrolnych mamy:

$$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$
$$C2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$
$$C4 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$
$$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Gdy teraz nowe bity kontrolne porównamy ze starymi, tworzymy słowo-syndrom:

	C8	C4	C2	C1
	0	1	1	1
$\oplus$	0	0	0	1
	0	1	1	0

Wynikiem jest 0110, co wskazuje, że pozycja bitowa 6, zawierająca bit danych 3, jest błędna.

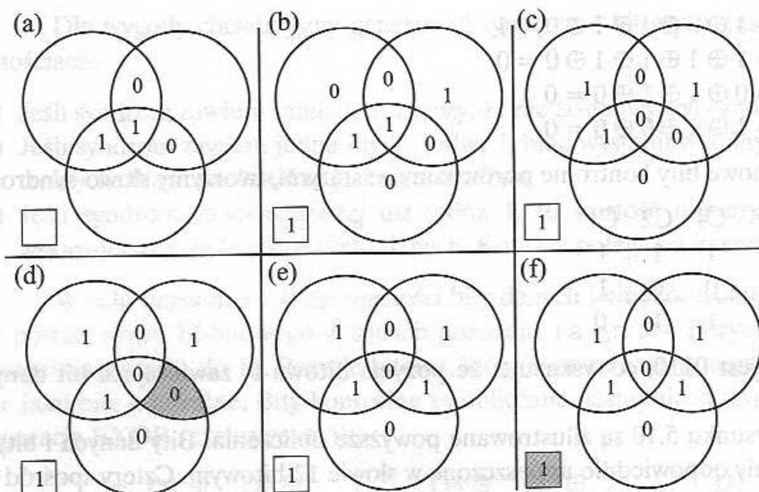
Na rysunku 5.10 są zilustrowane powyższe obliczenia. Bity danych i bity kontrolne zostały odpowiednio umieszczone w słowie 12-bitowym. Cztery spośród bitów danych mają wartość 1 (zacieniowane w tabeli), zaś wartości ich pozycji bitowych są poddawane operacji XOR w celu utworzenia kodu Hamminga 0111, który składa się z czterech cyfr kontrolnych. Cały zapisany blok to 001101001111. Załóżmy teraz, że bit danych 3 w pozycji bitowej 6 zawiera błąd i został zmieniony z 0 na 1. Powstał więc blok 001101101111. Wynikający stąd kod Hamminga nadal jest równy 0111. Operacja XOR zastosowana do kodu Hamminga i wszystkich wartości pozycji bitowych odpowiadających niezerowym bitom danych prowadzi do 0110. Wynik niezerowy oznacza wykrycie błędu i wskazuje, że błąd ten dotyczy bitu w pozycji 6.

Opisany właśnie kod jest znany jako *kod poprawiania pojedynczego błędu* (*single-error-correcting* – SEC). Częściej pamięć półprzewodnikowa jest wyposażana w kod poprawiania pojedynczego i wykrywania podwójnego błędu (SEC-DED). Jak widać w tabeli 5.2, takie kody wymagają jednego dodatkowego bitu w porównaniu z kodami SEC.

Na rysunku 5.11 jest pokazane, jak działa taki kod, znów dla przypadku 4-bitowego słowa danych. Przedstawiona sekwencja ujawnia, że jeśli występują dwa błędy

Pozycja bitowa	12	11	10	9	8	7	6	5	4	3	2	1
Numer pozycji	1100	1011	1010	1001	1000	0111	0110	1010	0100	0011	0010	0001
Bit danych	D8	D7	D6	D5		D4	D3	D2		D1		
Bit kontrolny					C8				C4		C2	C1
Słowo zapisane jako	0	0	1	1	0	1	0	0	1	1	1	1
Słowo pobrane jako	0	0	1	1	0	1	1	0	1	1	1	1
Numer pozycji	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit kontrolny					0				0		0	1

Rysunek 5.10. Obliczanie bitów kontrolnych



Rysunek 5.11. Kod Hamminga SEC-DED

(rys. 5.11c), procedura kontrolna jest błędna (rys. 5.11d) i pogarsza problem, tworząc trzeci błąd (rys. 5.11e). W celu pokonania tej trudności jest dodany ósmy bit, taki że całkowita liczba jedynek na wykresie jest parzysta. Dodatkowy bit parzystości umożliwia wychwycenie błędu (rys. 5.11f).

Kod korygowania błędów umożliwia poprawienie niezawodności pamięci kosztem zwiększenia jej złożoności. W przypadku jednobitowej organizacji mikroukładów kod SEC-DED jest na ogół uważany za odpowiedni. Na przykład w maszynach IBM 30*xx* dla każdych 64 bitów danych w pamięci głównej jest wykorzystywany 8-bitowy kod SEC-DED. W rezultacie rozmiar pamięci głównej jest o około 12% większy niż obserwowany przez użytkownika. W komputerach VAX jest wykorzystywany 7-bitowy kod SEC-DED dla każdych 32 bitów pamięci, co oznacza nadwyżkę 22%. W wielu współczesnych pamięciach DRAM dla każdych 128 bitów danych używanych jest 9 bitów kontrolnych, co oznacza nadwyżkę 7% [SHAR97].

5.3. Nowe rozwiązania organizacji DRAM

Jak mówiliśmy w rozdziale 2, jednym z najbardziej krytycznych wąskich gardeł systemu wykorzystującego procesory o wysokiej wydajności jest ich interfejs z wewnętrzną pamięcią główną. Interfejs ten jest najważniejszą ścieżką w całym systemie komputerowym. Podstawowym składnikiem pamięci głównej pozostaje mikroukład DRAM, tak jak był nim od dziesięcioleci. Od wczesnych lat siedemdziesiątych aż do teraz nie było znaczących zmian w architekturze DRAM. Tradycyjny mikroukład DRAM jest ograniczany zarówno przez swoją architekturę wewnętrzną, jak i przez swój interfejs z magistralą pamięciową procesora.

Widzieliśmy, że jednym ze sposobów zwiększenia wydajności pamięci głównej DRAM było wbudowanie jednej lub większej liczby szybkich pamięci podręcznych

SRAM między pamięcią główną DRAM a procesorem. Jednak pamięć SRAM jest o wiele bardziej kosztowna niż DRAM, zwiększanie zaś rozmiaru pamięci podręcznej powyżej pewnego poziomu przynosi malejące rezultaty.

W ciągu ostatnich kilku lat zbadano wiele możliwych ulepszeń podstawowej architektury DRAM, przy czym niektóre z nich znalazły się już na rynku. Dwoma rozwiązaniami, które dominują obecnie, są SDRAM i RDRAM. Pewną uwagę przyciągnęło również rozwiązanie CDRAM. Przeanalizujemy te rozwiązania w tym podrozdziale.

Synchroniczna pamięć DRAM

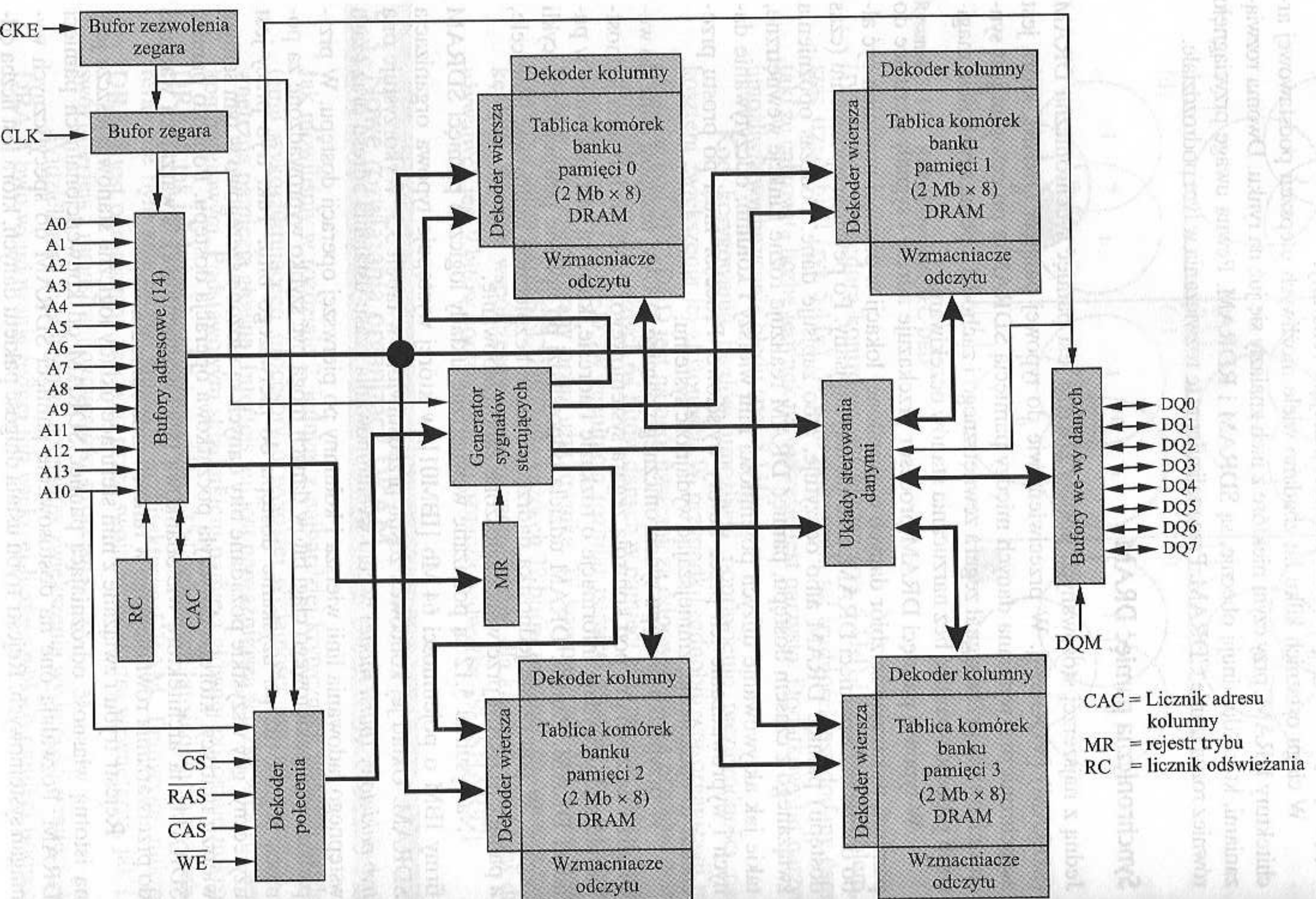
Jedną z najszerzej stosowanych postaci DRAM jest pamięć synchroniczna DRAM (SDRAM) [VOGL94]. W przeciwieństwie do typowej pamięci DRAM, która jest asynchroniczna, wymiana danych między pamięcią SDRAM a procesorem jest synchronizowana przez sygnał zegara zewnętrznego i zachodzi z pełną szybkością magistrali procesor-pamięć bez narzucania stanów oczekiwania.

W typowej pamięci DRAM procesor przekazuje adresy i sygnały sterujące do pamięci, wskazując że zbiór danych o określonej lokacji w pamięci powinien być albo odczytany z pamięci DRAM, albo do niej zapisany. Po pewnym opóźnieniu (czas dostępu) pamięć DRAM albo odczytuje, albo zapisuje dane. Podczas opóźnienia związanego z czasem dostępu, pamięć DRAM realizuje różne funkcje wewnętrzne, takie jak aktywowanie dużych pojemności linii wierszy i kolumn, odczytywanie danych i wyprowadzanie ich przez bufory wyjściowe. Procesor musi po prostu przeczekać to opóźnienie, zmniejszając wydajność systemu.

W przypadku dostępu synchronicznego pamięć DRAM przenosi dane do wewnątrz i na zewnątrz pod kontrolą zegara systemowego. Procesor lub inna jednostka nadrzędna wydaje informacje o rozkazie i adresie, które są zatrzaskiwane w pamięci DRAM. Pamięć DRAM udziela odpowiedzi po upływie pewnej liczby cykli zegara. W tym czasie jednostka nadrzędna może bezpiecznie realizować inne cele, a pamięć SDRAM przetwarza zgłoszone zapotrzebowanie.

Na rysunku 5.12 są pokazane wewnętrzne układy logiczne pamięci SDRAM firmy IBM o pojemności 64 Mb [IBM01], w której występuje typowa organizacja SDRAM. Układ jej końcówek został przedstawiony w tabeli 5.3. Wykorzystuje ona *tryb pakietowy* (*burst mode*) w celu wyeliminowania czasu ustalania adresu oraz czasu wstępnego ładowania linii wiersza i kolumny po pierwszej operacji dostępu. W przypadku trybu pakietowego ciągi bitów danych mogą być szybko wyprowadzone za pomocą zegara tuż po uzyskaniu dostępu do pierwszego bitu. Taki tryb pracy jest użyteczny, gdy wszystkie pożądane bity danych są ułożone szeregowo w tym samym wierszu matrycy, którego dotyczyła początkowa operacja dostępu. Ponadto pamięć SDRAM ma architekturę wielobankową (wielosegmentową), stwarzającą warunki do przetwarzania równoległego w ramach mikroukładu.

Rejestr trybu i związane z nim sterujące układy logiczne stanowią jeszcze jedną istotną własność odróżniającą pamięć SDRAM od konwencjonalnych pamięci DRAM. Pozwalają one na dostosowanie pamięci SDRAM do specyficznych wymagań systemowych. Rejestr trybu ustala długość pakietu danych, która jest liczbą od-



Rysunek 5.12. Synchroniczna, dynamiczna pamięć RAM (SDRAM)

Tabela 5.3. Układ końcówek pamięci SDRAM

A0 do A13	Wejścia adresu
CLK	Wejście zegara
CKE	Zezwolenie zegara
$\overline{CS}$	Wybór mikroukładu
$\overline{RAS}$	Bramkowanie adresu wiersza
$\overline{CAS}$	Bramkowanie adresu kolumny
$\overline{WE}$	Zezwolenie zapisu
DQ0 do DQ7	Wejście-wyjście danych
DWM	Maskowanie danych

dzielnych jednostek danych synchronicznie doprowadzanych do magistrali. Rejestr ten umożliwia także programiście regulowanie zwłoki między otrzymaniem zapotrzebowania na odczyt a rozpoczęciem transferu danych.

Pamięć SDRAM sprawuje się najlepiej, gdy dokonuje szeregowego transferu dużych bloków danych, co występuje w takich zastosowaniach, jak przetwarzanie tekstów, arkusze kalkulacyjne i multimedia.

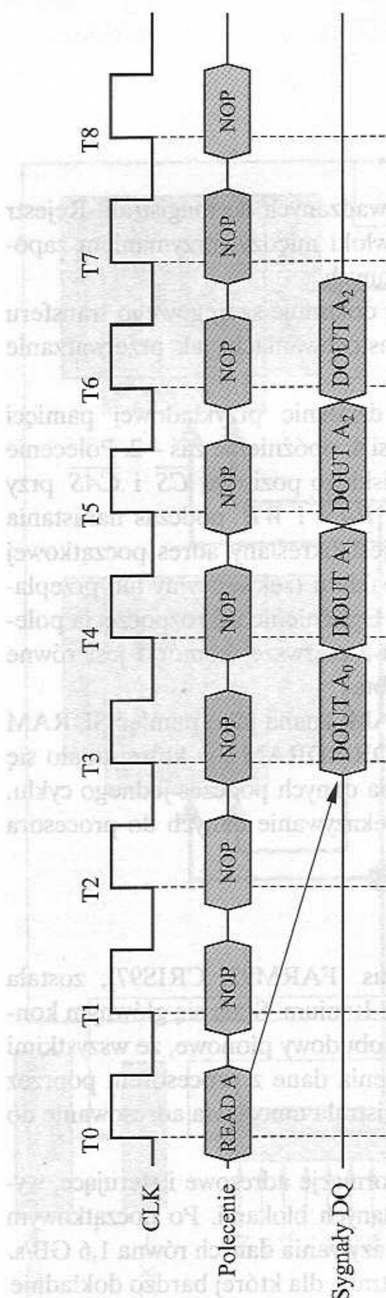
Na rysunku 5.13 zostało przedstawione działanie przykładowej pamięci SDRAM. W tym przypadku długość pakietu wynosi 4, opóźnienie zaś – 2. Polecenie odczytu pakietu jest inicjowane przez ustalenie niskiego poziomu $\overline{CS}$ i $\overline{CAS}$ przy jednoczesnym utrzymywaniu wysokiego poziomu $\overline{RAS}$ i $\overline{WE}$ podczas narastania impulsu zegarowego. Poprzez wejścia adresowe jest określany adres początkowej kolumny pakietu, zaś rejestr trybu ustala rodzaj pakietu (sekwencyjny lub przeplatany) i długość pakietu (1, 2, 4, 8 lub cała strona). Opóźnienie od rozpoczęcia polecenia do chwili ukazania się na wyjściach danych z pierwszej komórki jest równe wartości opóźnienia $\overline{CAS}$ ustalonej w rejestrze trybu.

Istnieje obecnie udoskonalona wersja SDRAM, znana jako pamięć SDRAM o podwójnej szybkości przekazywania danych (DDR-SDRAM), w której udało się ominąć ograniczenie jednorazowego przekazywania danych podczas jednego cyklu. Pamięć DDR-SDRAM umożliwia dwukrotne przekazywanie danych do procesora podczas jednego cyklu zegara.

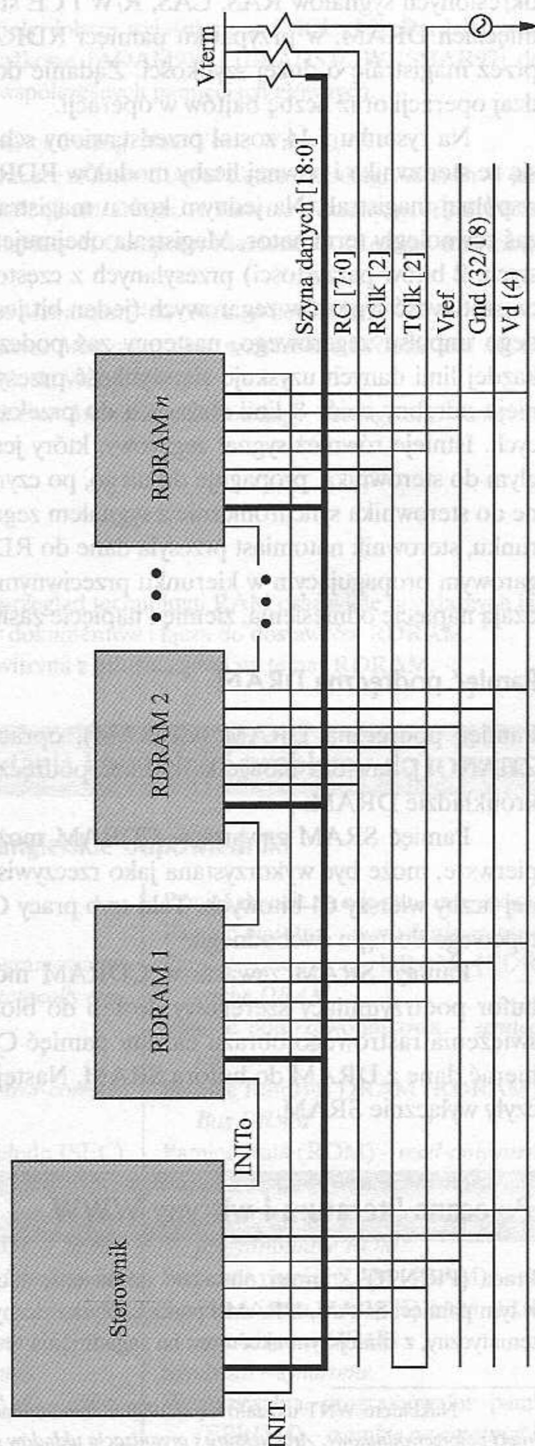
Pamięć RDRAM

Pamięć RDRAM, opracowana w firmie Rambus [FARM92, CRIS97], została przyjęta przez firmę Intel w procesorach Pentium i Itanium. Stała się głównym konkurentem SDRAM. Mikroukłady RDRAM mają obudowy pionowe, ze wszystkimi końcówkami po jednej stronie. Mikroukład wymienia dane z procesorem poprzez ponad 28 przewodów nie dłuższych od 12 cm. Magistrala umożliwia adresowanie do 320 mikroukładów RDRAM z szybkością 1,6 GB/s.

Specjalna magistrala RDRAM dostarcza informacje adresowe i sterujące, wykorzystując asynchroniczny protokół przesyłania danych blokami. Po początkowym czasie dostępu 480 ns osiągana jest szybkość przekazywania danych równa 1,6 GB/s. Tym, co umożliwia taką szybkość, jest sama magistrala, dla której bardzo dokładnie



Rysunek 5.13. Przebieg czasowy odczytu w pamięci SDRAM (długość pakietu = 4, opóźnienie $\overline{CAS}$ = 2)



Rysunek 5.14. Schemat pamięci RDRAM

określono impedancje, taktowanie i sygnały. Zamiast sterowania za pomocą jasno określonych sygnałów RAS, CAS, R/W i CE stosowanych w konwencjonalnych pamięciach DRAM, w przypadku pamięci RDRAM dostęp do danych zachodzi poprzez magistralę o dużej szybkości. Żądanie dostępu do danych zawiera adres, rodzaj operacji oraz liczbę bajtów w operacji.

Na rysunku 5.14 został przedstawiony schemat RDRAM. Konfiguracja składa się ze sterownika i pewnej liczby modułów RDRAM połączonych ze sobą za pomocą wspólnej magistrali. Na jednym końcu magistrali znajduje się sterownik, na drugim zaś równoległy terminator. Magistrala obejmuje 18 linii danych (16 danych rzeczywistych, 2 bitów parzystości) przesyłanych z częstotliwością dwukrotnie przekraczającą częstotliwość sygnałów zegarowych (jeden bit jest przesyłany podczas narastania każdego impulsu zegarowego, następny zaś podczas jego opadania). W rezultacie na każdej linii danych uzyskuje się szybkość przesyłania sygnałów równą 800 Mb/s. Istnieje odrębny zbiór 8 linii służących do przekazywania adresów i sygnałów sterujących. Istnieje również sygnał zegarowy, który jest wprowadzany na końcu przeciwległym do sterownika, propaguje do niego, po czym wraca. Moduł RDRAM wysyła dane do sterownika synchronicznie z sygnałem zegarowym propagującym w jednym kierunku, sterownik natomiast przesyła dane do RDRAM synchronicznie z sygnałem zegarowym propagującym w kierunku przeciwnym. Pozostałe linie magistrali doprowadzają napięcie odniesienia, ziemię i napięcie zasilania.

Pamięć podręczna DRAM

Pamięć podręczna DRAM (CDRAM), opracowana przez Mitsubishi [HIDA90, ZHAN01], zawiera niewielką pamięć podręczną SRAM (16 KB) w typowym mikroukładzie DRAM.

Pamięć SRAM zawarta w CDRAM może być używana na dwa sposoby. Po pierwsze, może być wykorzystana jako rzeczywista pamięć podręczna złożona z pewnej liczby wierszy 64-bitowych. Taki tryb pracy CDRAM jest efektywny w przypadku typowego dostępu swobodnego.

Pamięć SRAM zawarta w CDRAM może być także wykorzystywana jako bufor podtrzymujący szeregowy dostęp do bloku danych. Na przykład w celu odświeżenia rastrowego obrazu ekranu pamięć CDRAM może z wyprzedzeniem pobierać dane z DRAM do bufora SRAM. Następne operacje dostępu będą już dotyczyły wyłącznie SRAM.

5.4. Polecana literatura i witryny WWW

Praca [PRIN91]* stanowi obszerne omówienie technologii pamięci półprzewodnikowych, w tym pamięci SRAM, DRAM i pamięci błyskawicznych. [SHAR97] obejmuje taki sam zakres tematyczny, z silniejszym akcentem na zagadnienia testowania i niezawodności. [PRIN99] kon-

* Nakładem WNT ukazało się w 1999 r. tłumaczenie innej książki tej autorki pt. *Nowoczesne pamięci półprzewodnikowe. Architektura i organizacja układów pamięci DRAM i SRAM* (*przyp. red.*).

centruje się na zaawansowanych architekturach DRAM i SRAM. Głębsze ujęcie DRAM znajduje się w [KEET01].

Kody korekty błędów zostały dobrze wyjaśnione w [MCEL85]. Bardziej szczegółową analizę umożliwiają pozycje książkowe [ADAM91] i [BLAH83]. W [SHAR97] dokonano przeglądu kodów używanych we współczesnych pamięciach głównych.

ADAM91 Adamek J.: *Foundations of Coding*. New York, Wiley, 1991.

BLAH83 Blahut R.: *Theory and Practice of Error Control Codes*. Reading, Addison-Wesley, 1983.

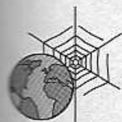
KEET01 Keeth B., Baker R.: *DRAM Circuit Design: A Tutorial.* Piscataway, IEEE Press, 2001.

MCEL85 McEliece R.: „The Reliability of Computer Memories". *Scientific American*, January 1985.

PRIN91 Prince B.: *Semiconductor Memories*. New York, Wiley, 1991.

PRIN99 Prince B.: *High Performance Memories: New Architecture DRAMs and SRAMs, Evolution and Function*. New York, Wiley, 1999.

SHAR97 Sharma A.: *Semiconductor Memories: Technology, Testing, and Reliability*. New York, IEEE Press, 1997.



Polecane witryny WWW:

- **The RAM Guide.** Doskonały przegląd technologii RAM oraz wiele przydatnych łącz.
- **RambusSite.** Użyteczny zbiór dokumentów i łącza do dostawców RDRAM.
- **RDRAM.** Kolejna użyteczna witryna z informacjami na temat RDRAM.

5.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Błąd przypadkowy – *soft error*
Błąd stały – *hard failure*
Elektrycznie wymazywalna, programowalna pamięć stała (EEPROM) – *electrically erasable programmable ROM*
Kod Hamminga – *Hamming code*
Kod korekcji błędów (EEC) – *error-correcting code*
Kod poprawiania pojedynczego błędu (SEC) – *single-error correcting code*
Kod poprawiania pojedynczego i wykrywania podwójnego błędu (SEC-DED) – *single-error-correcting, double-error-detecting code*
Korekcja błędów – *error correction*
Pamięć błyskawiczna – *flash memory*
Pamięć dynamiczna RAM (DRAM) – *dynamic RAM*
Pamięć głównie do odczytu – *read-mostly-memory*
Pamięć nielotna – *nonvolatile memory*
Pamięć podręczna DRAM (CDRAM) – *cache DRAM*
Pamięć półprzewodnikowa – *semiconductor memory*
Pamięć RamBus DRAM (RDRAM) – *RamBus DRAM*
Pamięć stała (ROM) – *read-only memory*
Pamięć ulotna – *volatile memory*
Programowalna pamięć stała (PROM) – *programmable ROM*
Statyczna pamięć RAM (SRAM) – *static RAM*
Synchroniczna pamięć DRAM (SDRAM) – *synchronous DRAM*
Syndrom – *syndrome*
Wymazywalna, programowalna pamięć stała (EPROM) – *erasable programmable ROM*

Pytania kontrolne

5.1. Jakie są podstawowe właściwości pamięci półprzewodnikowych?
5.2. Jakie są dwa znaczenia wyrażenia *pamięć o dostępie swobodnym*?
5.3. Jaka jest różnica między DRAM i SRAM pod względem zastosowań?
5.4. Jakie są różnice między DRAM i SRAM pod względem takich właściwości, jak szybkość, rozmiar i koszt?
5.5. Objaśnij, dlaczego jeden rodzaj pamięci RAM jest uważany za analogowy, inny zaś za cyfrowy.
5.6. Wymień niektóre zastosowania pamięci stałych.
5.7. Jakie są różnice między EPROM, EEPROM a pamięcią błyskawiczną?
5.8. Wyjaśnij funkcję wszystkich końcówek pokazanych na rys. 5.4b.
5.9. Co to jest bit parzystości?
5.10. Jak interpretuje się syndrom w kodzie Hamminga?
5.11. Czym różni się SDRAM od zwykłej pamięci DRAM?

Problemy do rozwiązania

5.1. Podaj przyczyny, dla których pamięci RAM tradycyjnie miały organizację 1-bitową, podczas gdy pamięci ROM – wielobitową.
5.2. Rozważ dynamiczną pamięć RAM, która wymaga cyklu odświeżania 64 razy w ciągu 1 ms. Każda operacja odświeżania wymaga 150 ns; cykl pamięci zajmuje 250 ns. Jaka część całego czasu działania pamięci musi być poświęcona na odświeżanie?
5.3. Zaprojektuj pamięć 16-bitową o pojemności całkowitej 8192 bit, wykorzystując mikroukłady SRAM 64 × 1 bit. Podaj konfigurację matrycy mikroukładów na płytce drukowanej, wskazując wszystkie wymagane sygnały wejściowe i wyjściowe, które umożliwiłyby przypisanie tej pamięci do najmniejszej przestrzeni adresowej. Projekt powinien pozwalać zarówno na dostęp do bajtów, jak i do słów 16-bitowych.
Źródło: [ALEX93].
5.4. W odniesieniu do kodu Hamminga pokazanego na rys. 5.10 przedstaw, co się stanie, jeśli błędny będzie nie bit danych, a bit kontrolny.
5.5. Załóżmy, że 8-bitowym słowem danych zapisanym w pamięci jest 11000010. Posługując się algorytmem Hamminga, określ bity kontrolne, jakie powinny być zapisane w pamięci wraz z tym słowem danych. Wykaż, że odpowiedź jest właściwa.
5.6. Dla słowa 8-bitowego bitami kontrolnymi powinny być 0111. Załóżmy, że dla pewnego słowa odczytywanego z pamięci obliczonymi bitami kontrolnymi są 1101. Jakie to jest słowo?
5.7. Ile bitów kontrolnych w kodzie korekcji błędów Hamminga jest wymaganych do wykrywania pojedynczych błędów w 1024-bitowym słowie danych?
5.8. Opracuj kod SEC dla 16-bitowego słowa danych. Utwórz ten kod dla słowa danych 0101000000111001. Wykaż, że kod ten prawidłowo zidentyfikuje błąd 5. bitu danych.
Źródło: [ALEX93].

Rozdział 6
Pamięć zewnętrzna

6.1. Dysk magnetyczny
- Magnetyczne mechanizmy odczytu i zapisu
- Organizacja i formatowanie danych
- Własności fizyczne
- Parametry wydajnościowe dysków

6.2. RAID
- RAID poziom 0
- RAID poziom 1
- RAID poziom 2
- RAID poziom 3
- RAID poziom 4
- RAID poziom 5
- RAID poziom 6

6.3. Pamięć optyczna
- Dysk kompaktowy
- Wszechstronny dysk cyfrowy

6.4. Taśma magnetyczna

6.5. Polecana literatura i witryny WWW

6.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
- Podstawowe terminy i ich angielskie odpowiedniki
- Pytania kontrolne
- Problemy do rozwiązania

Podstawowe spostrzeżenia

- Dyski magnetyczne pozostają najważniejszym składnikiem pamięci zewnętrznych. Zarówno dyski wymienne, jak i stałe (twarde) są powszechnie używane w systemach sięgających od komputerów osobistych poprzez duże (*mainframe*) do superkomputerów.
- W celu uzyskania większej wydajności i dostępności, rozwiązaniem popularnym w serwerach i dużych systemach jest technika dysków RAID. RAID oznacza rodzinę technik, w których wiele dysków używa się do równoległego przechowywania danych, przy czym do kompensowania uszkodzeń dysków służy wbudowana redundancja.
- Optyczna technologia przechowywania danych jest coraz ważniejsza we wszystkich rodzajach systemów komputerowych. Podczas gdy od wielu lat powszechnie stosowane były płyty CD-ROM, coraz większego znaczenia nabierają nowsze technologie, takie jak płyty CD i DVD umożliwiające zapis.

W tym rozdziale zajmiemy się urządzeniami i systemami pamięci zewnętrznej. Rozpoczniemy od najważniejszego urządzenia, jakim jest dysk magnetyczny. Dyski magnetyczne stanowią podstawę pamięci zewnętrznej praktycznie wszystkich systemów komputerowych. W następnym podrozdziale przeanalizujemy wykorzystanie tablic dysków w celu osiągnięcia większej wydajności, zwracając szczególną uwagę na rodzinę systemów zwanych RAID (*Redundant Array of Independent Disks*). Składnikiem wielu systemów komputerowych, którego znaczenie wzrasta, jest zewnętrzna pamięć optyczna – zajmiemy się nią w podrozdz. 6.3. Na zakończenie opiszemy magnetyczną pamięć taśmową.

6.1. Dysk magnetyczny

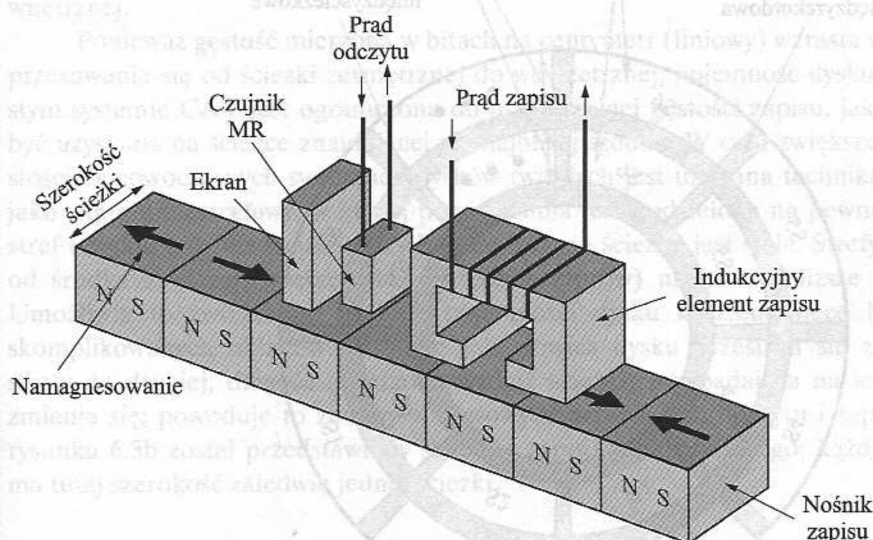
Dysk to okrągła płyta wykonana z materiału niemagnetycznego, zwana podłożem, pokryta materiałem magnetycznym. Tradycyjnie podłożem było aluminium lub jego stop. Niedawno zostały wprowadzone podłoża szklane. Mają one kilka zalet, do których należą:

- Zwiększenie jednorodności powierzchni warstwy magnetycznej; korzystnie wpływa to na niezawodność dysku.
- Znaczne zmniejszenie ogólnej liczby defektów powierzchni; redukuje to liczbę błędów odczytu i zapisu.
- Zdolność do obsługiwania mniejszych odstępów między głowicą a dyskiem.
- Większa sztywność ograniczająca zbędne ruchy dysku.
- Większa wytrzymałość na udary i uszkodzenia.

Magnetyczne mechanizmy odczytu i zapisu

Dane są zapisywane, a następnie odczytywane z dysku za pomocą przewodzącej cewki zwanej **głowicą**; w wielu systemach występują dwie oddzielne głowice do odczytu i zapisu. Podczas operacji odczytu lub zapisu głowica pozostaje nieruchoma, natomiast obraca się dysk.

Zapis polega na wykorzystaniu pola magnetycznego wytwarzanego przez prąd elektryczny płynący przez cewkę. Do głowicy są wysyłane impulsy, co powoduje zapisywanie wzorów magnetycznych na powierzchni znajdującej się pod głowicą, przy czym wzory te są różne dla prądów dodatnich i ujemnych. Sama głowica zapisu jest wykonana z materiału łatwo magnetyzującego się i ma kształt prostokątnego obwarzanka z przerwą z jednej strony, a z kilkoma zwojami drutu z drugiej (rys. 6.1). Prąd elektryczny w drucie indukuje pole magnetyczne w przerwie, co z kolei powoduje namagnesowanie niewielkiego obszaru nośnika. Odwrócenie kierunku przepływu prądu powoduje odwrócenie kierunku magnetyzacji nośnika.



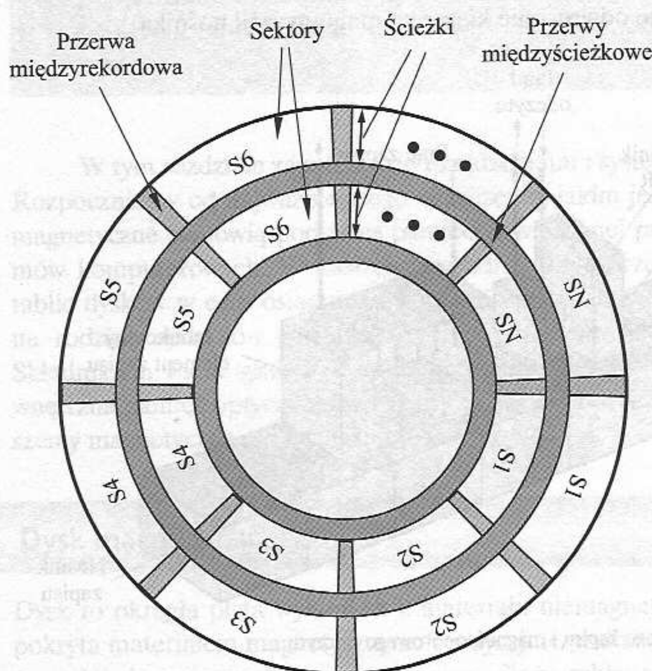
Rysunek 6.1. Głowice indukcyjnego zapisu i magnetooporowego odczytu

Tradycyjny mechanizm odczytu jest oparty na zjawisku indukowania prądu elektrycznego w cewce przez pole magnetyczne poruszające się względem tej cewki. Gdy powierzchnia dysku przesuwa się pod głowicą, generuje ona prąd o takiej samej biegunowości jak użyty do zapisu. Struktura głowicy odczytu jest w tym przypadku w zasadzie taka jak w przypadku zapisu, dlatego też ta sama głowica może służyć do obydwu celów. Tego rodzaju pojedyncze głowice są używane w systemach dyskietek i w starszych typach dysków twardych. We współczesnych systemach dysków twardych używany jest odmienny mechanizm odczytu wymagający oddzielnej głowicy odczytu, dla wygody umieszczonej obok głowicy zapisu. Głowica odczytu składa się z częściowo ekranowanego czujnika magnetooporowego (MR). Materiał

MR ma rezystancję elektryczną, która zależy od kierunku namagnesowania przesuwającego się pod nią nośnika. Poprzez przepuszczanie prądu przez czujnik MR, zmiany rezystancji są wykrywane jako sygnały napięciowe. Rozwiązanie MR umożliwia pracę z większymi częstotliwościami, co jest równoważne większym gęstościom zapisu i większym szybkościom działania.

Organizacja i formatowanie danych

Głowica jest względnie małym przyrządem umożliwiającym odczytywanie lub zapisywanie z części płyty obracającej się pod nią. To właśnie sprawiło, że organizacja danych na płycie ma postać koncentrycznego zespołu pierścieni, nazywanych **ścieżkami**. Każda ścieżka ma taką szerokość jak głowica. Na jednej powierzchni występują tysiące ścieżek.



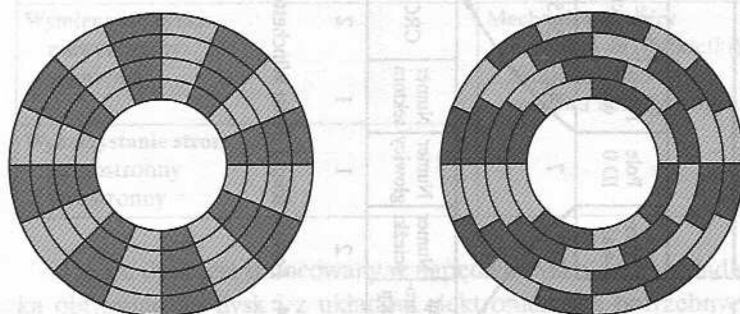
Rysunek 6.2. Rozkład danych na dysku

Ten rozkład danych jest przedstawiony na rys. 6.2. Sąsiednie ścieżki są oddzielone **przerwami**. Zapobiega to, a przynajmniej minimalizuje błędy spowodowane przez niewłaściwe ustawienie głowicy lub po prostu interferencję pola magnetycznego.

Dane są przenoszone na dysk i z dysku **sektorami** (rys. 6.2). Na ścieżkę przypadają zwykle setki sektorów, mogą one przy tym mieć długość ustaloną lub zmienną. W celu zapobieżenia nadmiernie wysokim wymaganiom dotyczącym precyzji systemu, sąsiednie sektory są oddzielone przerwami wewnątrzścieżkowymi (międzysektorowymi).

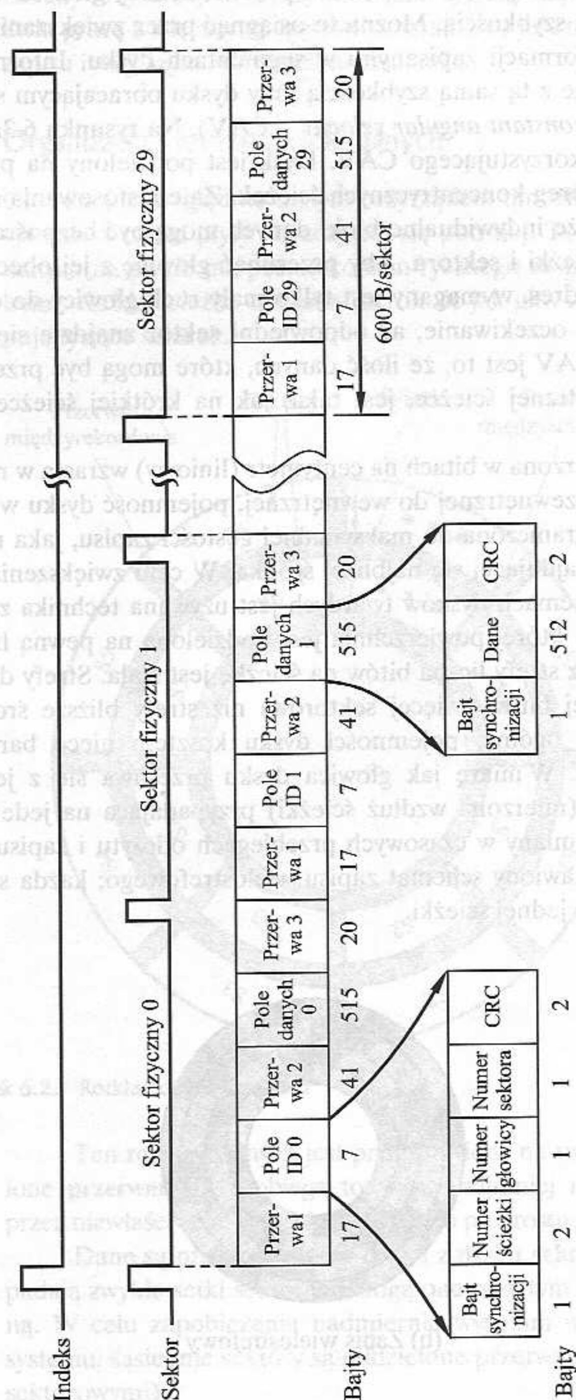
Bit położony bliżej środka obracającego się dysku porusza się względem ustalonego punktu (np. głowicy odczytu-zapisu) wolniej niż bit na obrzeżu płyty, trzeba więc znaleźć sposób kompensowania zmian prędkości, żeby głowica mogła odczytywać dane z tą samą szybkością. Można to osiągnąć przez zwiększanie odległości między bitami informacji zapisanymi w segmentach dysku. Informacje mogą więc być odczytywane z tą samą szybkością przy dysku obracającym się ze **stałą prędkością kątową** (*constant angular velocity* – CAV). Na rysunku 6.3a jest pokazany układ dysku wykorzystującego CAV. Dysk jest podzielony na pewną liczbę sektorów oraz na szereg koncentrycznych ścieżek. Zaletą stosowania stałej prędkości kątowej jest to, że indywidualne bloki danych mogą być bezpośrednio adresowane za pomocą ścieżki i sektora. Aby przesunąć głowicę z jej obecnego położenia pod określony adres, wymagany jest tylko mały ruch głowicy do określonej ścieżki oraz krótkie oczekiwanie, aż odpowiedni sektor znajdzie się pod głowicą. Wadą napędów CAV jest to, że ilość danych, które mogą być przechowywane na długiej, zewnętrznej ścieżce, jest taka, jak na krótkiej ścieżce wewnętrznej.

Ponieważ **gęstość** mierzona w bitach na centymetr (liniowy) wzrasta w miarę przesuwania się od ścieżki zewnętrznej do wewnętrznej, pojemność dysku w prostym systemie CAV jest ograniczona do maksymalnej gęstości zapisu, jaka może być uzyskana na ścieżce znajdującej się najbliżej środka. W celu zwiększenia gęstości, w nowoczesnych systemach dysków twardych jest używana technika znana jako **zapis wielostrefowy**, w której powierzchnia jest podzielona na pewną liczbę stref (zwykle 16). Wewnątrz strefy liczba bitów na ścieżkę jest stała. Strefy dalsze od środka zawierają więcej bitów (więcej sektorów) niż strefy bliższe środka. Umożliwia to zwiększenie ogólnej pojemności dysku kosztem nieco bardziej skomplikowanych układów. W miarę jak głowica dysku przesuwa się z jednej strefy do drugiej, długość (mierzona wzdłuż ścieżki) przypadająca na jeden bit zmienia się; powoduje to zmiany w czasowych przebiegach odczytu i zapisu. Na rysunku 6.3b został przedstawiony schemat zapisu wielostrefowego; każda strefa ma tutaj szerokość zaledwie jednej ścieżki.



(a) Stała prędkość kątowa (b) Zapis wielostrefowy

Rysunek 6.3. Schematy zapisu na dysku



Rysunek 6.4. Format ścieżki dysku Winchester (Seagate ST506)

W jaki sposób są ustalane miejsca sektorów w ramach ścieżki? Musi oczywiście istnieć pewien punkt startowy na ścieżce oraz sposób identyfikowania początku i końca każdego sektora. Wymagania te są spełniane za pomocą danych kontrolnych zapisanych na dysku. Dysk jest więc formatowany za pomocą dodatkowych danych wykorzystywanych tylko przez napęd dysku i niedostępnych dla użytkownika.

Przykład sformatowania dysku jest pokazany na rys. 6.4. W tym przypadku każda ścieżka zawiera 30 sektorów o ustalonej długości, po 600 bajtów każdy. Każdy sektor mieści 512 bajtów danych oraz informacje kontrolne wykorzystywane przez sterownik dysku. Pole ID jest unikatowym (jednoznacznym) identyfikatorem lub adresem wykorzystywanym do lokalizowania określonego sektora. Bajt SYNCH jest specjalnym wzorem bitowym wyznaczającym początek pola. Numer ścieżki identyfikuje ścieżkę na powierzchni. Numer głowicy identyfikuje głowicę, ponieważ dysk ma zwykle wiele powierzchni (będzie to wyjaśnione w dalszym ciągu). Zarówno pola ID, jak i pola danych zawierają kod służący do wykrywania błędów.

Własności fizyczne

W tabeli 6.1 są wymienione główne własności, które umożliwiają różnicowanie poszczególnych rodzajów dysków magnetycznych. Po pierwsze, głowica może być nieruchoma lub poruszać się wzdłuż promienia płyty. W przypadku dysków z **nieruchomą głowicą**, występuje jedna głowica zapisu-odczytu na jedną ścieżkę. Wszystkie głowice są zmontowane na sztywnym ramieniu, które rozciąga się przez wszystkie ścieżki; tego rodzaju systemy stanowią obecnie rzadkość. W przypadku dysków z **ruchomą głowicą**, istnieje tylko jedna głowica zapisu-odczytu. Jak poprzednio, głowica jest umocowana na ramieniu. Ponieważ jednak musi istnieć możliwość pozycjonowania głowicy nad dowolną ścieżką, ramię może być w tym celu wydłużane lub skracane.

Tabela 6.1. Własności systemów dyskowych

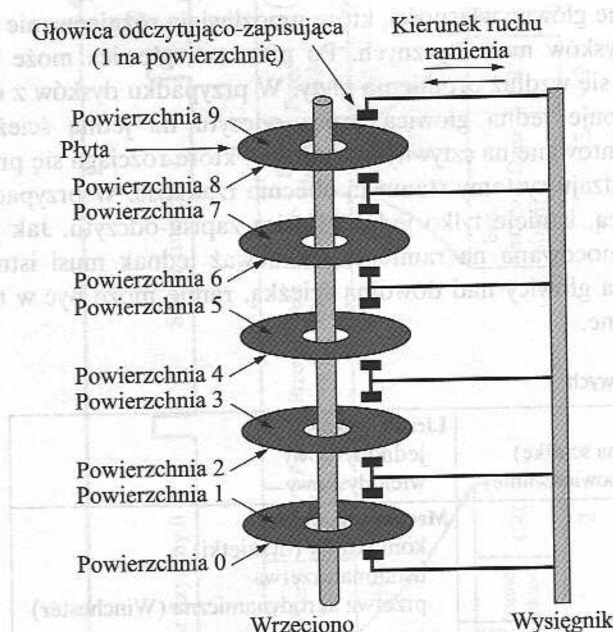
Ruch głowicy głowica nieruchoma (jedna na ścieżkę) głowica ruchoma (jedna na powierzchnię)	**Liczba dysków** jednodyskowy wielodyskowy
Wymienność dysku niewymienny wymienny	**Mechanizm głowicy** kontaktowy (dyskietki) ustalona przerwa przerwa aerodynamiczna (Winchester)
Wykorzystanie stron jednostronny dwustronny	

Sam dysk jest umocowany w napędzie dysku, który składa się z ramienia, wałka obracającego dysk i z układów elektronicznych potrzebnych do wprowadzania i wyprowadzania danych binarnych. Dysk **niewymienny** jest na stałe mocowany w napędzie dysku. Dysk **wymienny** może być usunięty i zastąpiony innym dyskiem. Zaletą tego ostatniego jest nieograniczona ilość danych osiągalna przy ograniczonej

liczbie systemów dyskowych. Ponadto dysk taki może być przenoszony z jednego systemu komputerowego do innego. Dyskietki i dyski kasetowe ZIP stanowią przykłady dysków wymiennych.

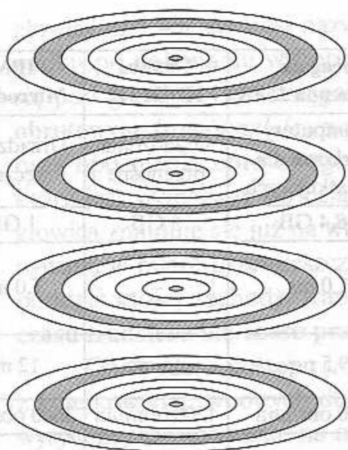
W przypadku większości dysków warstwa magnetyczna jest nanoszona po obu stronach płyty; mówimy wtedy o **dysku dwustronnym**. Niektóre tańsze systemy dyskowe wykorzystują **dyski jednostronne**.

Niektóre napędy dysków umożliwiają stosowanie wielu płyt ustawionych pionowo w odległościach ułamka cala. Wykorzystuje się wówczas wiele ramion (rys. 6.5). W dyskach wielopłytowych używa się głowicy ruchomej, przy czym jedna głowica odczytu-zapisu przypada na jedną powierzchnię płyty. Wszystkie głowice są umocowane mechanicznie w taki sposób, że znajdują się w jednakowej odległości od środka dysku i poruszają się razem. Dlatego w każdej chwili wszystkie głowice znajdują się nad ścieżkami, których odległość od środka dysku jest jednakowa. Zbiór wszystkich ścieżek znajdujących się w takim samym miejscu płyty jest określany jako **cylinder**. Na przykład wszystkie cieniowane ścieżki na rys. 6.6 należą do jednego cylindra.



Rysunek 6.5. Składniki napędu dyskowego

Wreszcie, mechanizm głowicy stanowi podstawę do klarownej klasyfikacji dysków na trzy rodzaje. Tradycyjnie głowica odczytu-zapisu była umieszczana w ustalonej odległości nad płytą, przy czym była pozostawiona przerwa powietrzna. Krańcowo różniące się rozwiązanie mechanizmu głowicy polega na pozostawieniu jej w fizycznym kontakcie z płytą podczas operacji odczytu lub zapisu. Mechanizm



Rysunek 6.6. Ścieżki i cylindry

ten jest stosowany w przypadku napędu **dyskietek**, które są niewielkimi, elastycznymi płytkami i stanowią najtańszy rodzaj dysków magnetycznych.

Aby zrozumieć powód powstania trzeciego rodzaju dysków, musimy skomentować zależność między gęstością danych a rozmiarem przerwy powietrznej. Głowica musi generować lub wykrywać pole elektromagnetyczne o wielkości wystarczającej do zapisu i odczytu. Im węższa jest głowica, tym bardziej musi być zbliżona do powierzchni płyty, aby urządzenie działało. Ponieważ węższa głowica oznacza węższe ścieżki i dzięki temu większą gęstość danych, jest to pożądane. Jednak im bliżej znajduje się głowica w stosunku do dysku, tym większe jest ryzyko błędu spowodowanego przez zanieczyszczenia lub niedokładności. W wyniku ulepszenia technologii wprowadzono dysk typu Winchester. Głowice dysku Winchester pracują w zamkniętych zespołach napędowych, które są niemal wolne od zanieczyszczeń. Zostały zaprojektowane do działania w mniejszej odległości od powierzchni dysków w porównaniu z konwencjonalnymi głowicami dysków, co umożliwia większą gęstość upakowania danych. Głowica ta to w rzeczywistości aerodynamiczny pasek folii, spoczywający lekko na powierzchni płyty w czasie, gdy dysk jest nieruchomy. Ciśnienie powietrza generowane przez wirujący dysk wystarcza, aby spowodować uniesienie się folii nad powierzchnią. Dzięki temu można wykorzystać węższe głowice, które pracują bliżej powierzchni płyty niż konwencjonalne sztywne głowice dysków[1].

W tabeli 6.2 zostały zestawione parametry typowych, współczesnych dysków o dużej wydajności.

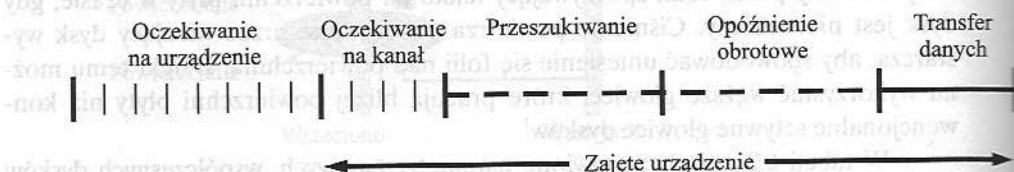
[1] Historycznie rzecz biorąc, określenie *Winchester* zostało początkowo użyte przez IBM jako nazwa kodowa modelu dysku 3340 przed jego zaanonsowaniem. Model 3340 był wymiennym pakietem dysków z głowicami zamkniętymi wewnątrz pakietu. Termin ten jest obecnie stosowany w odniesieniu do dowolnego zamkniętego napędu dysków wykorzystującego głowice aerodynamiczne. Dysk Winchester jest powszechnie stosowany w komputerach osobistych i w stacjach roboczych, gdzie jest określany jako dysk twardy.

Tabela 6.2. Typowe parametry dysków twardych

Własności	Seagate Barracuda 180	Seagate Cheetah X15-36LP	Seagate Barracuda 36ES	Toshiba HDD1242	IBM Microdrive
Zastosowanie	Serwery o dużej pojemności	Serwery o dużej wydajności	Komputery biurkowe dla początkujących	Urządzenia przenośne	Urządzenia ręczne
Pojemność	181,6 GB	36,7 GB	18,4 GB	5 GB	1 GB
Minimalny czas przeszukiwania od ścieżki do ścieżki	0,8 ms	0,3 ms	1,0 ms	–	1,0 ms
Średni czas przeszukiwania	7,4 ms	3,6 ms	9,5 ms	15 ms	12 ms
Prędkość wrzeciona	7200 obr/min	15 K obr/min	7200 obr/min	4200 obr/min	3600 obr/min
Średnie opóźnienie obrotowe	4,17 ms	2 ms	4,17 ms	7,14 ms	8,33 ms
Maksymalna szybkość transferu	160 MB/s	522–709 MB/s	25 MB/s	66 MB/s	13,3 MB/s
Bajtów na sektor	512	512	512	512	512
Sektorów na ścieżkę	793	485	600	63	–
Ścieżek na cylinder (liczba powierzchni płyt)	24	8	2	2	2
Cylindrów (liczba ścieżek po jednej stronie płyty)	24 247	18 479	29 851	10 350	–

Parametry wydajnościowe dysków

Szczegóły działania dysku w kategoriach wejścia-wyjścia zależą od systemu komputerowego, systemu operacyjnego oraz od sprzętowego rozwiązania kanału wejścia-wyjścia i sterownika dysku. Ogólny schemat przebiegów czasowych transferu danych z dysku został pokazany na rys. 6.7.



Rysunek 6.7. Przebiegi czasowe transferu danych z dysku

Podczas pracy dysk obraca się ze stałą prędkością. W celu zapisu lub odczytu głowica musi być ustawiona nad pożądaną ścieżką i na początku pożądanego sektora na tej ścieżce. Wybór ścieżki polega na przesunięciu głowicy w przypadku systemu z ruchomą głowicą lub na elektronicznym wyborze jednej głowicy w systemie z nieruchomymi głowicami. W systemie z ruchomą głowicą czas pozycjonowania

głowicy nad ścieżką nosi nazwę **czasu przeszukiwania** (*seek time*). W każdym przypadku po dokonaniu wyboru ścieżki system czeka, aż odpowiedni sektor znajdzie się pod głowicą. Czas osiągnięcia głowicy przez sektor jest nazywany **opóźnieniem obrotowym** (*rotational latency*). Suma czasu przeszukiwania, jeśli taki występuje, oraz opóźnienia obrotowego nazywa się **czasem dostępu** (*access time*) – czasem, który jest wymagany do osiągnięcia stanu umożliwiającego odczyt lub zapis. Gdy głowica znajduje się już na właściwej pozycji, operacja odczytu lub zapisu jest dokonywana w miarę przemieszczania się sektora pod głowicą; jest to ta część operacji, podczas której zachodzi transfer danych. Czas wymagany do transferu nosi nazwę **czasu transferu** lub **czasu przesyłania** danych.

Obok czasu dostępu i czasu transferu występuje kilka opóźnień wynikających z kolejkowania, typowych dla dyskowych operacji wejścia-wyjścia. Gdy jakiś proces wysyła zapotrzebowanie na operację wejścia-wyjścia, musi najpierw poczekać w kolejce, aż dane urządzenie będzie dostępne. Z tą chwilą urządzenie to zostaje przypisane do procesu. Jeśli urządzenie dzieli pojedynczy kanał wejścia-wyjścia lub zbiór takich kanałów z innymi napędami dyskowymi, może wystąpić dodatkowe oczekiwanie, aż będzie dostępny kanał. W tym momencie jest realizowane przeszukiwanie inicjujące dostęp do dysku.

W niektórych systemach o najwyższych parametrach przeznaczonych dla serwerów stosuje się technikę znaną jako odczytywanie obrotowo-pozycyjne (*rotational positional sensing* – RPS). Oto jak ono funkcjonuje. Gdy zostanie wydane polecenie przeszukiwania, kanał jest uwalniany dla umożliwienia realizacji innych operacji wejścia-wyjścia. Po zakończeniu przeszukiwania urządzenie określa, kiedy dane znajdą się pod głowicą. Gdy określony sektor zbliży się do głowicy, urządzenie próbuje wznowić połączenie z hostem. Jeśli albo jednostka sterująca, albo kanał są zajęte, próba wznowienia połączenie nie udaje się i urządzenie musi wykonać pełny obrót, zanim znów spróbuje je wznowić, co nosi nazwę *chybienia RPS*. Jest to dodatkowe opóźnienie, które musi być dodane do przebiegu czasowego pokazanego na rys. 6.7.

Czas przeszukiwania

Czas przeszukiwania to czas wymagany do przeniesienia ramienia dysku na wymaganą ścieżkę. Okazuje się, że jest on trudny do wyznaczenia ilościowego. Składa się on z dwóch podstawowych składników: czasu rozruchu oraz czasu, jaki zajmuje przejście przez ścieżki, które muszą być przekroczone, zanim ramię dostosuje się do prędkości. Niestety, ten czas przekraczania ścieżek nie jest liniową funkcją ich liczby, lecz obejmuje czas rozruchu i czas ustalania położenia (czas po znalezieniu się głowicy nad ścieżką docelową do chwili, w której zostanie potwierdzona identyfikacja ścieżki).

Znaczna poprawa tego czasu wynikła z zastosowania mniejszych i lżejszych składników napędu. Przed laty typowy dysk miał średnicę 36 cm, podczas gdy obecnie najpopularniejszą średnicą jest 8,9 cm, co zmniejsza odległości, jakie musi przebywać ramię. Typowy średni czas przeszukiwania współczesnych dysków twardych nie przekracza 10 ms.

Opóźnienie obrotowe

Dyski (z wyłączeniem dyskietek) obracają się z prędkością od 3600 obr/min (w przypadku urządzeń przenoszonych w ręku, takich jak cyfrowe aparaty fotograficzne) do – podczas pisania tej książki – 15 000 obr/min. Przy tej najwyższej prędkości obrót następuje co 4 ms. Średnio opóźnienie obrotowe wynosi wówczas 2 ms. Dyskietki obracają się zwykle z prędkościami od 300 do 600 obr/min. Zatem średnie opóźnienie obrotowe wynosi od 100 do 50 ms.

Czas transferu

Czas transferu (przesyłania) danych z dysku lub na dysk zależy od prędkości obrotowej dysku w sposób następujący:

$$T = \frac{b}{rN}$$

gdzie:
T – czas transferu,
b – liczba bajtów, jakie są przesyłane,
N – liczba bajtów na ścieżce,
r – prędkość obrotowa w obr/min.

Zatem łączny średni czas dostępu może być wyrażony jako

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

gdzie T_s jest średnim czasem przeszukiwania. Pamiętajmy, że w napędzie strefowym liczba bajtów w ścieżce jest zmienna; komplikuje to obliczenia.

Porównanie przebiegów czasowych

Po zdefiniowaniu powyższych parametrów przeanalizujmy dwie różne operacje wejścia-wyjścia w celu ukazania niebezpieczeństwa polegania na wartościach średnich. Rozważmy dysk z reklamowanym przeciętnym czasem przeszukiwania 4 ms, prędkością obrotową 7500 obr/min oraz z 512-bajtowymi sektorami przy 500 sektorach na ścieżkę. Załóżmy, że chcemy odczytać plik składający się z 2500 sektorów, łącznie 1,28 MB. Chcielibyśmy oszacować łączny czas transferu.

Załóżmy po pierwsze, że plik jest zapisany na dysku w sposób maksymalnie zwarty. Oznacza to, że plik zajmuje wszystkie sektory 5 sąsiadujących ze sobą ścieżek (5 ścieżek × 500 sektorów na ścieżkę = 2500 sektorów). Nosi to nazwę *sekwencyjnej organizacji danych*. Czas odczytania pierwszej ścieżki wynosi więc

Średni czas przeszukiwania	4 ms
Opóźnienie obrotowe	4 ms
Odczyt 500 sektorów	8 ms
	16 ms

Załóżmy, że pozostałe ścieżki mogą być teraz odczytane bez dodatkowego czasu przeszukiwania. Oznacza to, że operacja wejścia-wyjścia może nadążać z przepływem danych z dysku. Co najwyżej musimy więc się liczyć z opóźnieniem obrotowym w odniesieniu do kolejnych ścieżek. Zatem każda kolejna ścieżka zostanie odczytana w ciągu 4 + 8 = 12 ms. Czas odczytania całego pliku:

Czas łączny = 16 + 4 × 12 ms = 64 ms = 0,064 s

Obliczmy teraz czas potrzebny do odczytania tych samych danych przy użyciu dostępu bezpośredniego w miejsce sekwencyjnego. Oznacza to, że sektory są rozproszone przypadkowo po całym dysku. W odniesieniu do każdego sektora

Średni czas przeszukiwania	4	ms
Opóźnienie obrotowe	4	ms
Odczyt 1 sektora	0,016	ms
	8,016	ms

Czas łączny = 2500 × 8,016 = 20,04 s

Jest oczywiste, że kolejność odczytywania sektorów z dysku ma ogromny wpływ na wydajność operacji wejścia-wyjścia. W przypadku operacji dostępu do plików, w których jest odczytywanych lub zapisywanych wiele sektorów, mamy pewną kontrolę nad sposobem rozmieszczenia sektorów danych, co zostanie omówione w następnym rozdziale. Jednak nawet w przypadku sięgania do plików w środowisku wieloprogramowym, o ten sam dysk będą rywalizowały różne zapotrzebowania na operacje wejścia-wyjścia. Warto wiec przeanalizować sposoby, dzięki którym można byłoby zwiększyć wydajność dyskowych operacji wejścia-wyjścia w stosunku do realizowanych w warunkach całkowicie bezpośredniego dostępu do dysku. Prowadzi to do rozważań o algorytmach szeregowania dysków, co jest rolą systemu operacyjnego i wykracza poza zakres tej książki (omówienie tych zagadnień znajduje się w [STAL01]).

6.2. RAID

Jak mówiliśmy wcześniej, wydajność pamięci pomocniczych rosła wyraźnie wolniej niż wydajność procesorów i pamięci głównych. To niedopasowanie spowodowało, że system pamięci dyskowych stał się być może głównym ogniskiem zainteresowania w pracach nad poprawą ogólnej wydajności systemów komputerowych.

Podobnie jak w przypadku pozostałych obszarów wydajności komputera, projektanci pamięci dyskowych stwierdzili, że przy ograniczonej możliwości ulepszenia pojedynczych składników dodatkowe zwiększenie wydajności można osiągnąć, wykorzystując równolegle wiele składników. W przypadku pamięci dyskowej doprowadziło to do opracowania tablic dysków, które pracują niezależnie i równo-

legle. Gdy mamy do czynienia z wieloma dyskami, oddzielne zapotrzebowania na operacje wejścia-wyjścia mogą być przetwarzane równolegle, jeśli tylko poszukiwany blok danych jest rozproszony na wielu dyskach.

W przypadku używania wielu dysków istnieje wiele sposobów organizowania danych i zwiększenia niezawodności przez wykorzystanie nadmiarowości (redundancji). Mogłoby to utrudnić opracowanie schematów baz danych, które mogłyby być używane na wielu platformach sprzętowych i z różnymi systemami operacyjnymi. Na szczęście w przemyśle uzgodniono znormalizowany schemat projektowania baz danych dla pamięci wielodyskowych, znany jako *redundancyjna tablica niezależnych dysków* – RAID (*Redundant Array of Independent Disks*). Schemat RAID składa się z siedmiu poziomów[2], od zerowego do piątego. Poziomy te nie implikują zależności hierarchicznej, lecz wyznaczają różne architektury, które mają trzy cechy wspólne:

1. RAID jest zespołem fizycznie istniejących napędów dyskowych widzianych przez system operacyjny jako pojedynczy napęd logiczny.
2. Dane są rozproszone w tych napędach, tworzących tablicę.
3. Redundancyjna pojemność dysków jest wykorzystywana do przechowywania informacji o parzystości; gwarantuje to odzyskiwanie danych w przypadku uszkodzenia dysku.

Szczegółowe rozwiązania dotyczące drugiej i trzeciej cechy są różne dla różnych poziomów RAID. Poziom RAID 0 nie obsługuje trzeciej cechy.

Termin RAID został po raz pierwszy zaproponowany w artykule napisanym przez grupę badaczy z University of California w Berkeley [PATT88][3]. W artykule przedstawiono różne konfiguracje i zastosowania RAID, a także wprowadzono definicję poziomów RAID stosowaną do dzisiaj. Strategia RAID polega na zastąpieniu napędów dyskowych o wielkiej pojemności wieloma napędami o mniejszej pojemności i na rozmieszczeniu danych w taki sposób, aby umożliwić jednoczesny dostęp do danych w wielu napędach. Prowadzi to do zwiększenia wydajności operacji wejścia-wyjścia i ułatwia stopniowe zwiększanie pojemności.

Unikatowym wkładem wniesionym w propozycji RAID jest efektywne zwiększenie niezawodności przez redundancję. Chociaż jednoczesne działanie wielu głowic i urządzeń uruchamiających umożliwiło przyspieszenie działania wejścia-wyjścia, jednak stosowanie wielu podzespołów zwiększa prawdopodobieństwo uszkodzenia.

---

[2] Niektórzy badacze i niektóre firmy wprowadziły dodatkowe poziomy, jednak powszechne uzgodnienie obejmuje tylko siedem poziomów opisanych w tym podrozdziale.

[3] W artykule tym akronim RAID dotyczył określenia Redundant Array of Inexpensive Disks (redundancyjna matryca niedrogich dysków). Termin „niedrogich" zastosowano dla odróżnienia małych, stosunkowo tanich dysków w matrycy RAID od stanowiących alternatywę, pojedynczych, dużych i drogich dysków (SLED). Dzisiaj SLED należy do przeszłości, a podobna technologia dysków jest stosowana zarówno w konfiguracjach RAID, jak i w pozostałych. Wobec tego w przemyśle przyjęto termin „niezależnych" w celu podkreślenia, że matryca RAID umożliwia znaczącą poprawę wydajności i niezawodności.

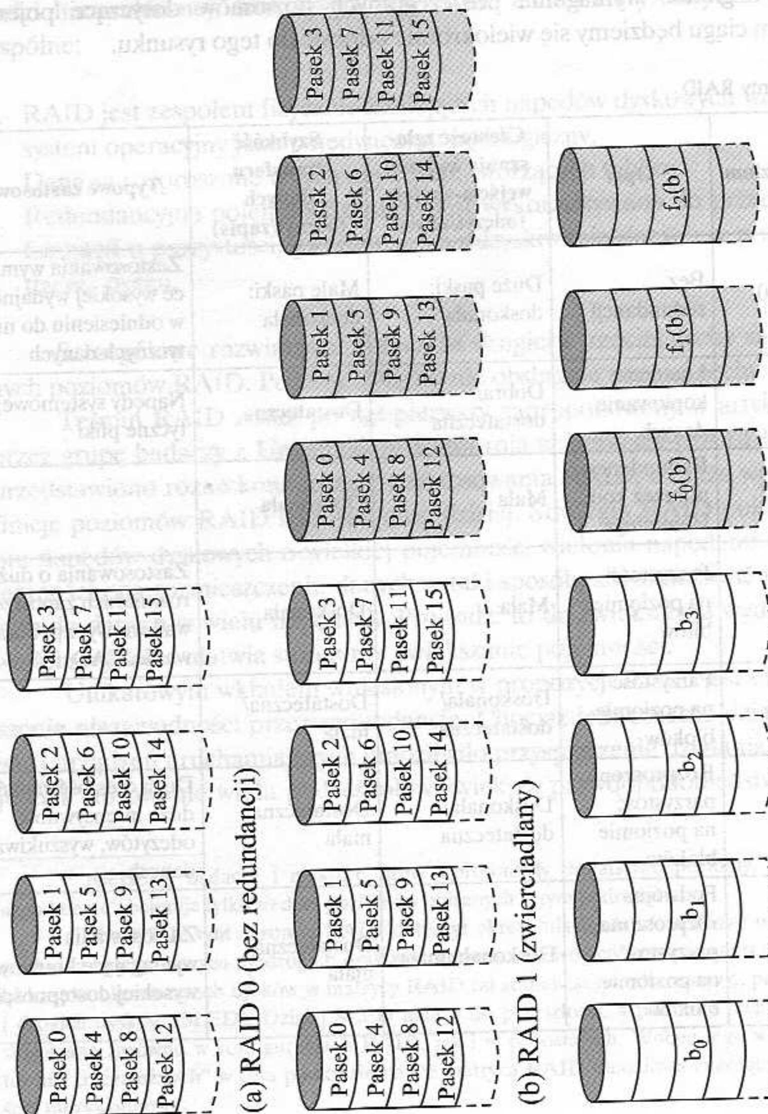
Aby skompensować tę zmniejszoną niezawodność, RAID wykorzystuje informację o parzystości, która pozwala na odtwarzanie danych utraconych z powodu uszkodzenia dysku.

Przeanalizujemy teraz każdy z poziomów RAID. W tabeli 6.3 są podsumowane własności wszystkich siedmiu poziomów. Spośród nich poziomy 2 i 4 nie są oferowane komercyjnie, a ich akceptacja w przemyśle nie jest prawdopodobna. Mimo to opis tych poziomów pomaga wyjaśnić decyzje projektowe odnoszące się do pozostałych poziomów.

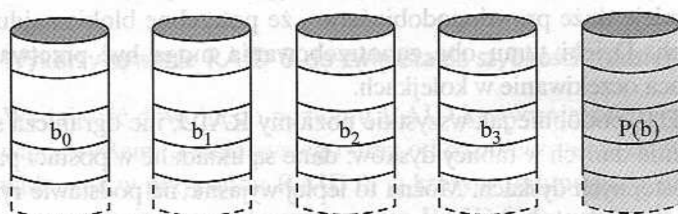
Na rysunku 6.8 został przedstawiony przykład użycia siedmiu poziomów RAID obsługujących dane wymagające czterech dysków (bez uwzględniania redundancji). Na rysunku pokazano układ danych użytkownika i danych redundancyjnych, a także ukazano względne wymagania poszczególnych poziomów dotyczące pojemności. W dalszym ciągu będziemy się wielokrotnie odnosić do tego rysunku.

Tabela 6.3. Poziomy RAID

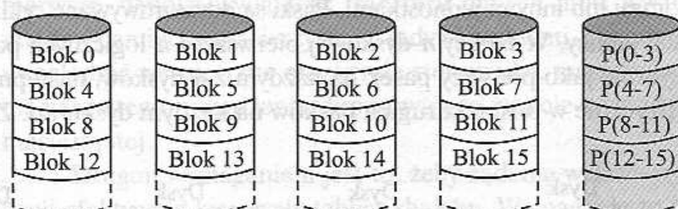
Kategoria	**Poziom**	**Opis**	**Częstość zgłaszania żądania wejścia-wyjścia (odczyt/zapis)**	**Szybkość transferu danych (odczyt/zapis)**	**Typowe zastosowanie**
Paskowanie	0	Bez redundancji	Duże paski: doskonała	Małe paski: doskonała	Zastosowania wymagające wysokiej wydajności w odniesieniu do niekrytycznych danych
Kopiowanie lustrzane	1	Lustrzane kopiowanie danych	Dobra/ dostateczna	Dostateczna	Napędy systemowe; krytyczne pliki
Dostęp równoległy	2	Redundancja poprzez kod Hamminga	Mała	Doskonała	
	3	Parzystość na poziomie bitów	Mała	Doskonała	Zastosowania o dużych rozmiarach zapotrzebowań we-wy, np. obrazowanie, CAD
Dostęp niezależny	4	Parzystość na poziomie bloków	Doskonała/ dostateczna	Dostateczna/ mała	
	5	Rozproszona parzystość na poziomie bloków	Doskonała/ dostateczna	Dostateczna/ mała	Duża częstość żądań, duża intensywność odczytów, wyszukiwanie danych
	6	Podwójna rozproszona parzystość na poziomie bloków	Doskonała/mała	Dostateczna/ mała	Zastosowania wymagające krańcowo wysokiej dostępności



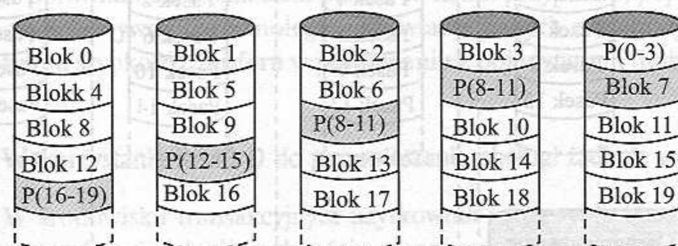
Rysunek 6.8. Poziomy RAID



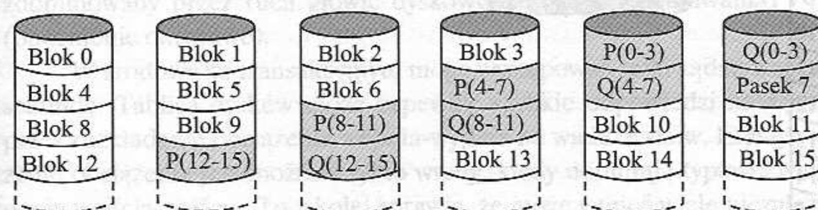
(d) RAID 3 (parzystość z przeplataniem bitów)



(e) RAID 4 (parzystość na poziomie bloków)



(f) RAID 5 (rozproszona parzystość na poziomie bloków)



(g) RAID 6 (podwójna redundancja)

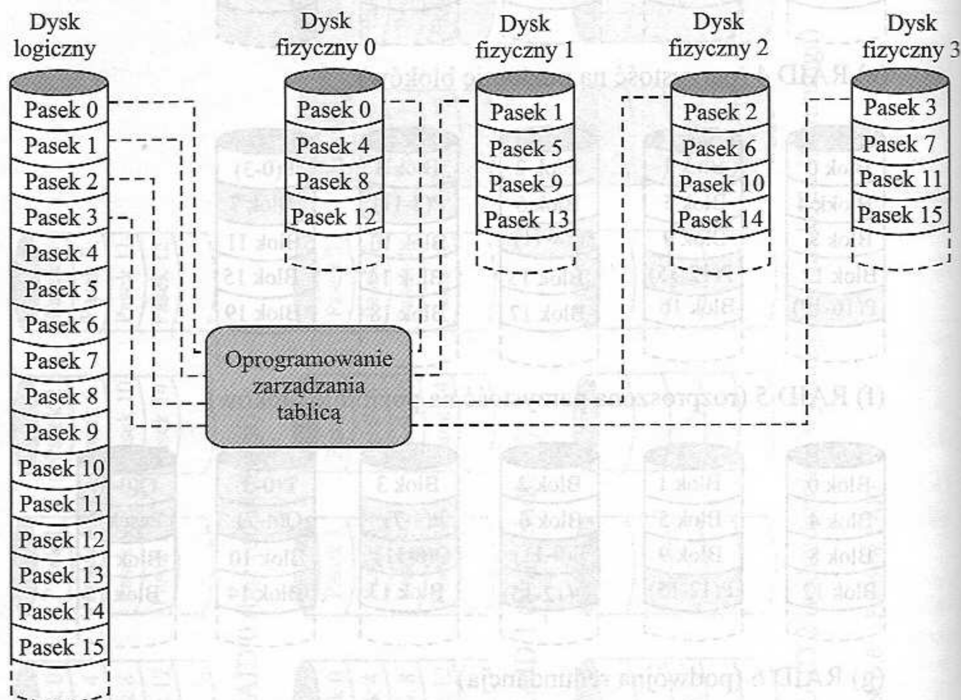
Rysunek 6.8. Poziomy RAID (cd.)

RAID poziom 0

RAID 0 nie powinien być zaliczany do rodziny RAID, ponieważ w celu zwiększenia wydajności nie przewiduje on redundancji. Istnieją jednak nieliczne zastosowania (należą do nich niektóre superkomputery), w których wydajność i pojemność są najważniejsze, a niski koszt jest ważniejszy od zwiększonej niezawodności.

W przypadku RAID 0 dane użytkownika i dane systemowe są rozproszone na wszystkich dyskach tablicy. Stanowi to wyraźną korzyść w stosunku do dużego pojedynczego dysku: jeśli dwa różne żądania wejścia-wyjścia dotyczą dwóch różnych bloków danych, istnieje duże prawdopodobieństwo, że potrzebne bloki znajdują się na różnych dyskach. Dzięki temu oba zapotrzebowania mogą być przetwarzane równolegle, co skraca oczekiwanie w kolejkach.

Jednak RAID 0, podobnie jak wszystkie poziomy RAID, nie ogranicza się do prostego rozproszenia danych w tablicy dysków: dane są układane w postaci pasków (ang. *striped*) na dostępnych dyskach. Można to lepiej wyjaśnić na podstawie rys. 6.9. Wszystkie dane użytkownika i systemowe mogą być postrzegane jako przechowywane na jednym dysku logicznym. Dysk jest podzielony na paski; paski te mogą być fizycznymi blokami, sektorami lub innymi jednostkami. Paski są odwzorowywane cyklicznie na kolejnych dyskach tablicy. W tablicy n-dyskowej pierwszych n logicznych pasków przechowuje się fizycznie jako pierwszy pasek na każdym z n dysków, następnych n pasków mieści się fizycznie w postaci drugich pasków na każdym dysku itd. Zaletą



Rysunek 6.9. Odwzorowanie danych dla poziomu RAID 0

takiego rozkładu jest to, że jeśli pojedyncze zapotrzebowanie wejścia-wyjścia dotyczy wielu logicznie sąsiadujących pasków, to nawet *n* pasków dotyczących tego zapotrzebowania może być obsługiwanych równolegle, co znacznie redukuje czas transferu wejścia-wyjścia.

Na rysunku 6.9 jest pokazane wykorzystanie oprogramowania zarządzania tablicą w celu odwzorowania między logiczną a fizyczną przestrzenią dysków. Oprogramowanie to może być uruchamiane albo w podsystemie dyskowym, albo w komputerze macierzystym.

Wykorzystywanie RAID 0 do zwiększania szybkości transferu danych

Wydajność dowolnego poziomu RAID krytycznie zależy od rozkładu zapotrzebowania systemu macierzystego oraz od rozmieszczenia danych. Jest to najwyraźniej widoczne w przypadku RAID 0, w którym redundancja nie zaburza analizy. Rozpatrzmy najpierw wykorzystywanie RAID 0 do osiągania dużych szybkości przesyłania danych. W takich zastosowaniach muszą być spełnione dwa wymagania. Po pierwsze, wysoka wydajność transferu danych musi występować na całej drodze między pamięcią macierzystą a indywidualnymi napędami dysków. Należy więc uwzględnić wewnętrzne magistrale sterownika, magistrale wejścia-wyjścia systemu macierzystego, urządzenia dopasowujące wejście-wyjście oraz magistrale pamięci macierzystej.

Drugim wymaganiem jest to, żeby żądania wejścia-wyjścia pochodzące z aplikacji efektywnie kierowały tablicą dysków. Wymaganie to jest spełnione, jeśli typowe żądanie dotyczy takiej ilości logicznie sąsiadujących danych, która jest duża w porównaniu z rozmiarem paska. W takim przypadku, pojedyncze żądanie wejścia-wyjścia wywołuje równoległe przesyłanie danych z wielu dysków, zwiększając efektywną szybkość transferu w porównaniu z pojedynczym dyskiem.

Wykorzystanie RAID 0 do przyspieszenia obsługi żądania wejścia-wyjścia

W środowisku transakcyjnym użytkownik jest zwykle bardziej zainteresowany czasem odpowiedzi niż szybkością transferu. W przypadku indywidualnego żądania wejścia-wyjścia dotyczącego małej ilości danych, czas dostępu wejścia-wyjścia jest zdominowany przez ruch głowic dyskowych (czas przeszukiwania) i ruch dysków (opóźnienie obrotowe).

W środowisku transakcyjnym mogą występować setki żądań wejścia-wyjścia na sekundę. Tablica dysków może zapewnić szybkie odpowiedzi na żądanie dostępu przez rozkładanie obciążenia wejścia-wyjścia na wiele dysków. Efektywne równoważenie obciążenia jest możliwe tylko wtedy, kiedy dominują typowe, wielokrotne żądania wejścia-wyjścia. To z kolei sprawia, że może istnieć wiele niezależnych aplikacji lub że pojedyncza aplikacja o charakterze transakcyjnym może powodować wielokrotne, asynchroniczne żądanie wejścia-wyjścia. Na wydajność ma również wpływ rozmiar paska. Jeśli jest on stosunkowo duży, w wyniku czego pojedyncze żądanie

wejścia-wyjścia wywołuje tylko pojedynczy dostęp do dysku, to wiele oczekujących żądań wejścia-wyjścia można realizować równolegle, co redukuje czas oczekiwania w kolejce każdego żądania.

RAID poziom 1

RAID 1 różni się od poziomów RAID od 2 do 6 sposobem osiągania redundancji. W pozostałych schematach RAID jest wykorzystywana pewna forma obliczeń parzystości w celu wprowadzenia redundancji. Natomiast w przypadku RAID 1 redundancja jest osiągana po prostu przez powielenie wszystkich danych. Jak widać na rys. 6.8b, wykorzystywane jest paskowanie danych, podobnie jak w RAID 0. Jednak w tym przypadku każdy pasek logiczny jest odwzorowywany na dwóch oddzielnych dyskach fizycznych, dzięki czemu każdy dysk w tablicy ma swój dysk zwierciadlany zawierający te same dane.

Organizacja RAID 1 ma wiele pozytywnych aspektów:

1. Żądanie odczytu może być obsługiwane przez ten spośród dwóch dysków zawierających potrzebne dane, który wymaga mniejszego czasu przeszukiwania plus opóźnienie obrotowe.
2. Żądanie zapisu wymaga aktualizacji obu odpowiednich pasków, jednak może to być wykonywane równolegle. Wobec tego wydajność zapisu jest dyktowana przez wolniejszy z dwóch zapisów (tzn. ten, który zajmuje dłuższy czas przeszukiwania plus opóźnienie obrotowe). W przypadku RAID 1 nie występuje jednak „kara" związana z zapisem. W RAID 2 do 6 są wykorzystywane bity parzystości. Jeśli więc jest aktualizowany pojedynczy pasek, oprogramowanie zarządzania tablicą musi obliczyć i zaktualizować bity parzystości oprócz aktualizowania samego paska danych.
3. W razie awarii rozwiązanie jest proste. Gdy napęd ulega uszkodzeniu, dane mogą być uzyskane z drugiego napędu.

Główną wadą RAID 1 jest koszt; wymaga on 2-krotnie większej przestrzeni dyskowej w porównaniu z dyskiem logicznym, który obsługuje. Z tego powodu wykorzystanie konfiguracji RAID 1 pozostaje raczej ograniczone do napędów przechowujących oprogramowanie systemowe oraz inne bardzo ważne pliki. W tych przypadkach RAID 1 zapewnia rezerwowe dane dostępne w czasie realnym, dzięki czemu w razie uszkodzenia dysku wszystkie ważne dane są natychmiast dostępne.

W środowisku transakcyjnym RAID 1 umożliwia szybkie uzyskiwanie odpowiedzi na żądania wejścia-wyjścia, jeśli tylko główna część żądań dotyczy odczytu. W takiej sytuacji wydajność RAID 1 może być niemal 2-krotnie większa niż wydajność RAID 0. Jeśli jednak istotna część żądań wejścia-wyjścia dotyczy zapisu, to może nie występować znaczna poprawa wydajności w stosunku do RAID 0. RAID 1 może także mieć większą wydajność niż RAID 0 w przypadku zastosowań wymagających intensywnego transferu danych przy znaczącym udziale odczytów. Wzrost wydajności następuje, gdy możliwe jest rozdzielenie każdego żądania odczytu w ten sposób, aby w operacji uczestniczyły oba dyski.

RAID poziom 2

RAID 2 i 3 wykorzystują metodę dostępu równoległego. W przypadku tablicy o dostępie równoległym wszystkie dyski uczestniczą w realizacji każdego żądania wejścia-wyjścia. Zwykle poszczególne napędy są synchronizowane tak, że w dowolnym momencie każda głowica znajduje się w tej samej pozycji nad każdym dyskiem.

Podobnie jak w pozostałych schematach RAID, wykorzystywane jest paskowanie danych. W RAID 2 i 3 paski są bardzo małe, często równe jednemu bajtowi lub słowu. W RAID 2 kod korekcji błędów jest obliczany na podstawie odpowiednich bitów na każdym dysku danych, a bity kodu są przechowywane w odpowiednich pozycjach bitowych zlokalizowanych na wielu dyskach parzystości. Zwykle jest wykorzystywany kod Hamminga, który umożliwia korygowanie błędów jednobitowych i wykrywanie dwubitowych.

Chociaż RAID 2 wymaga mniejszej liczby dysków niż RAID 1, nadal jest raczej kosztowny. Liczba dysków redundancyjnych jest proporcjonalna do logarytmu liczby dysków danych. W przypadku pojedynczego odczytu następuje jednoczesny dostęp do wszystkich dysków. Potrzebne dane oraz towarzyszący im kod korekcji błędów są dostarczane do sterownika tablicy. Jeśli wystąpił błąd jednobitowy, sterownik może go rozpoznać i natychmiast skorygować, dzięki czemu czas dostępu odczytu nie ulega wydłużeniu. W przypadku pojedynczego zapisu musi nastąpić dostęp do wszystkich dysków danych i dysków parzystości.

Poziom RAID 2 byłby właściwym wyborem tylko w środowisku, w którym występuje wiele błędów dyskowych. Jeśli indywidualne dyski i napędy dyskowe wyróżniają się dużą niezawodnością, RAID 2 jest rozwiązaniem rozrzutnym i po prostu nie jest wykorzystywany.

RAID poziom 3

RAID 3 jest zorganizowany podobnie do RAID 2. Różnica polega na tym, że wymaga on tylko jednego dysku redundancyjnego, niezależnie od wielkości matrycy dysków. W RAID 3 wykorzystuje się dostęp równoległy, przy czym dane są rozmieszczone w postaci małych pasków. Zamiast kodu korekcyjnego jest obliczany bit parzystości dla zespołu indywidualnych bitów znajdujących się w tej samej pozycji na wszystkich dyskach danych.

Redundancja

W przypadku uszkodzenia napędu sięga się do napędu parzystości, po czym następuje rekonstrukcja danych na podstawie danych zawartych w pozostałych urządzeniach. Gdy uszkodzony dysk jest wymieniony, zaginione dane mogą być do niego wprowadzone, a przerwana operacja może być podjęta na nowo.

Rekonstrukcja danych jest całkiem prosta. Rozważmy tablicę złożoną z pięciu napędów, w której dyski X0÷X3 zawierają dane, natomiast X4 jest dyskiem parzystości. Parzystość i-tego bitu jest obliczana następująco:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

Załóżmy, że został uszkodzony napęd X1. Jeśli dodamy $X4(i) \oplus X1(i)$ do obu stron powyższego równania, to otrzymamy

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

Wobec tego zawartość dowolnego paska danych na dowolnym dysku danych w tablicy może być zregenerowana na podstawie zawartości odpowiednich pasków na pozostałych dyskach tablicy. Zasada ta jest prawdziwa dla poziomów RAID 3, 4, 5 i 6.

W przypadku uszkodzenia dysku wszystkie dane są nadal dostępne w tzw. trybie zredukowanym. W tym trybie dla odczytu zaginione dane są regenerowane na bieżąco za pomocą obliczeń (jak wyżej). Gdy dane są zapisywane w zredukowanej tablicy RAID 3, musi być zachowywana zgodność parzystości w celu umożliwienia późniejszej regeneracji. Powrót do pełnego działania wymaga wymiany uszkodzonego dysku i zregenerowania całej jego zawartości na nowym dysku.

Wydajność

Ponieważ dane są rozmieszczone w bardzo małych paskach, RAID 3 może osiągać bardzo duże szybkości transferu danych. Jakiekolwiek żądanie wejścia-wyjścia spowoduje równoległe przesyłanie danych ze wszystkich dysków danych. Poprawa wydajności jest szczególnie widoczna w przypadku dużych transferów.

Należy jednak zauważyć, że w określonym momencie może być realizowane tylko jedno żądanie wejścia-wyjścia. Wobec tego w środowisku transakcyjnym wydajność ulega pogorszeniu.

RAID poziom 4

W RAID 4 do 6 wykorzystuje się metodę dostępu niezależnego. W tablicy o dostępie niezależnym każdy dysk działa niezależnie, dzięki czemu oddzielne żądania wejścia-wyjścia mogą być obsługiwane równolegle. Z tego względu tablice o dostępie niezależnym są bardziej odpowiednie w przypadku zastosowań wymagających szybkiej odpowiedzi na żądania wejścia-wyjścia, są natomiast stosunkowo mało przydatne w zastosowaniach wymagających dużych szybkości transferu danych.

Podobnie jak w przypadku pozostałych poziomów RAID, w RAID 4 wykorzystuje się paskowanie danych. Paski są tu stosunkowo duże. Pasek parzystości tworzony bit po bicie jest obliczany na podstawie odpowiednich pasków na każdym dysku danych, a bity parzystości są przechowywane w odpowiednim pasku na dysku parzystości.

W schemacie RAID 4 występuje pogorszenie wydajności zapisu, jeśli realizowane jest żądanie zapisu małej ilości danych. Za każdym razem, gdy następuje zapis, oprogramowanie zarządzania tablicą musi zaktualizować nie tylko dane użytkownika, ale również odpowiednie bity parzystości. Rozważmy tablicę złożoną z pięciu napędów, w której dyski X0÷X3 zawierają dane, X4 natomiast jest dyskiem parzystości. Załóżmy, że dokonywany jest zapis obejmujący tylko pasek na dysku X1. Początkowo dla każdego i-tego bitu zachodzi następująca zależność:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

Jeśli potencjalnie zmienione bity oznaczymy primem, to po aktualizacji zachodzi

$$X4'(i) = X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) =$$
$$= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) =$$
$$= X4(i) \oplus X1(i) \oplus X1'(i)$$

W celu obliczenia nowego bitu parzystości oprogramowanie zarządzania tablicą musi odczytać stary pasek użytkownika i stary pasek parzystości. Następnie musi ono zaktualizować te oba paski za pomocą nowych danych oraz na nowo obliczonej parzystości. Tak więc każdy zapis paska powoduje dwa odczyty i dwa zapisy.

W przypadku żądania zapisu dużej ilości danych, angażującego paski na wszystkich dyskach, parzystość jest z łatwością obliczana wyłącznie na podstawie nowych bitów danych. Dzięki temu dysk parzystości może być aktualizowany równolegle z dyskami danych i nie występują dodatkowe odczyty ani zapisy.

Przy każdej operacji zapisu muszą być zaktualizowane dane na dysku parzystości, który przez to może się stać wąskim gardłem.

RAID poziom 5

RAID 5 jest zorganizowany podobnie jak RAID 4. Różnica polega na tym, że w przypadku RAID 5 paski parzystości są rozproszone na wszystkich dyskach. Typowo wykorzystuje się schemat cykliczny (*round-robin*), pokazany na rys. 6.8f. W przypadku tablicy n-dyskowej pasek parzystości jest umieszczany na różnych dyskach w odniesieniu do pierwszych n pasków danych, po czym schemat ten jest powtarzany.

Rozproszenie pasków parzystości na wszystkich napędach zapobiega ewentualnemu występowaniu wąskich gardeł, o których była mowa w odniesieniu do RAID 4.

RAID poziom 6

RAID 6 został przedstawiony w kolejnym artykule badaczy z Berkeley [KATZ89]. W tym rozwiązaniu są przeprowadzane dwa różne obliczenia parzystości, ich zaś wyniki są zapisywane w oddzielnych blokach na różnych dyskach. Wobec tego tablica RAID 6, w której dane użytkownika wymagają N dysków, składa się z $N + 2$ dysków.

Rozwiązanie to zostało przedstawione na rys. 6.8g. P i Q są dwoma różnymi algorytmami kontroli danych. Jednym z nich jest obliczanie XOR zastosowane w RAID 4 i 5. Drugi jednak jest niezależnym algorytmem sprawdzania danych. Umożliwia to regenerowanie danych nawet wówczas, gdy dwa dyski zawierają błędne dane użytkownika.

Zaletą rozwiązania RAID 6 jest to, że zapewnia ono ekstremalnie wysoką dostępność danych. W ramach przeciętnego czasu między naprawami (*mean time to repair* – MTTR) musiałyby ulec uszkodzeniu trzy dyski, aby dane zostały utracone. Z drugiej strony rozwiązaniu temu towarzyszy istotne pogorszenie warunków zapisu, ponieważ każdy zapis wpływa na obydwa bloki parzystości.

6.3. Pamięć optyczna

W roku 1983 wprowadzono jeden z najbardziej udanych wyrobów konsumpcyjnych wszystkich czasów: cyfrowy system akustyczny wykorzystujący płyty kompaktowe (CD). CD jest niewymazywalnym dyskiem, na którym można zapisać ponad 60 minut informacji audio na jednej stronie. Ogromny sukces komercyjny płyty CD umożliwił opracowanie taniej technologii optycznego dysku pamięciowego, która stanowi zapowiedź rewolucji obejmującej przechowywanie danych w komputerach. W ciągu minionych kilku lat wprowadzono wiele systemów dysków optycznych (tabela 6.4). Krótko przeanalizujemy każdy z nich.

Tabela 6.4. Dyski optyczne

CD	Dysk kompaktowy. Niewymazywalny dysk do przechowywania informacji audio w postaci cyfrowej. W systemie znormalizowanym są używane dyski o średnicy 12 cm, przy czym czas ciągłego odtwarzania przekracza 60 min.
CD-ROM	Pamięć stała na dysku kompaktowym. Niewymazywalny dysk służący do przechowywania danych komputerowych. W systemie znormalizowanym są używane dyski o średnicy 12 cm i pojemności ponad 650 MB.
CD-R	Zapisywalny dysk kompaktowy. Podobny do CD-ROM. Użytkownik może zapisać dysk tylko jednorazowo.
CD-RW	Wielokrotnie zapisywalny dysk kompaktowy. Podobny do CD-ROM. Użytkownik może wielokrotnie wymazywać i ponownie zapisywać dysk.
DVD	Wszechstronny dysk cyfrowy, zwany również cyfrowym dyskiem wideo. Technika cyfrowego, skompresowanego zapisu informacji wideo, a także wielkich ilości innych danych cyfrowych. Średnice używanych dysków to 8 i 12 cm. Przy dwustronnym zapisie mają one pojemność sięgającą 17 GB. Podstawowa płyta DVD służy tylko do odczytu (DVD-ROM).
DVD-R	Zapisywalny dysk DVD. Podobny do DVD-ROM. Użytkownik może zapisywać dysk tylko jednorazowo. Do tego celu mogą służyć tylko dyski jednostronne.
DVD-RW	Wielokrotnie zapisywalny dysk DVD. Podobny do DVD-ROM. Użytkownik może wielokrotnie zapisywać dysk. Do tego celu mogą służyć tylko dyski jednostronne.

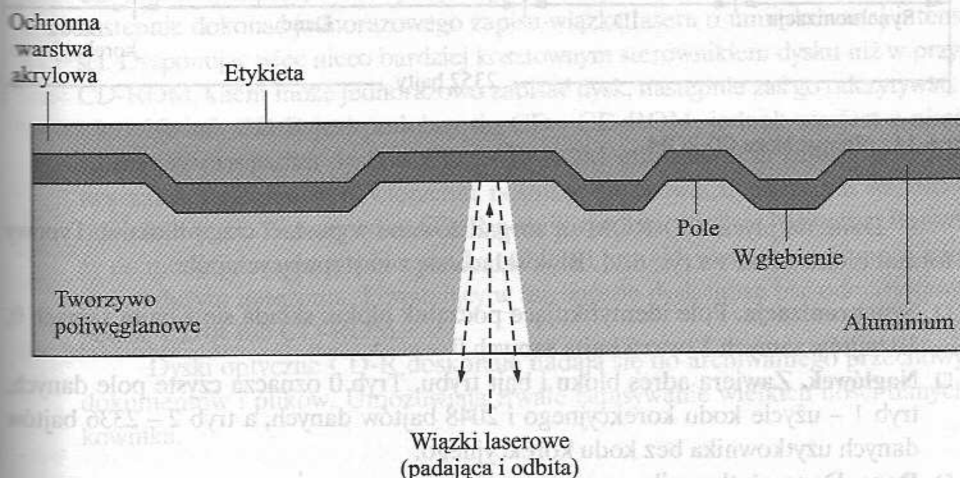
Dysk kompaktowy

CD-ROM

Zarówno akustyczne płyty CD, jak i płyty CD-ROM (*compact disk read-only memory* – pamięć stała na dyskach kompaktowych) są oparte na podobnej technologii. Główną różnicę stanowi to, że odtwarzacze CD-ROM mają podzespoły korekcyjne

w celu poprawnego transferu danych z dysku do komputera. Oba typy dysków są produkowane w ten sam sposób. Dysk jest wykonywany z żywicy, np. poliwęglanowej. Informacja zarejestrowana cyfrowo (albo muzyka, albo dane komputerowe) jest nanoszona w postaci mikroskopijnych zagłębień na powierzchnię odbijającą. Po raz pierwszy wykonuje się to za pomocą dobrze zogniskowanego światła lasera o dużej mocy, dzięki czemu powstaje dysk wzorcowy. Dysk ten służy do wykonywania formy, która służy do tłoczenia kopii na dyskach poliwęglanowych. Powierzchnia z naniesioną informacją jest następnie pokrywana warstwą o dużym współczynniku odbicia światła, zwykle aluminiową lub złotą. Z kolei warstwa ta jest chroniona przed kurzem i zarysowaniem za pomocą przezroczystego lakieru akrylowego. Na zakończenie metodą sitodruku nanosi się etykietę.

Informacja jest odczytywana z płyty CD lub z CD-ROM za pomocą lasera małej mocy wbudowanego w odtwarzacz płyt lub napęd CD-ROM. Laser świeci przez przezroczyste pokrycie, a dysk jest wprawiany w ruch obrotowy przez silnik (rys. 6.10). Natężenie odbitego światła zmienia się, gdy napotyka ono wgłębienie. Dokładniej rzecz biorąc, gdy wiązka laserowa napotyka wgłębienie, światło ulega rozproszeniu i z powrotem w stronę źródła trafia światło o mniejszym natężeniu. Obszary między wgłębieniami noszą nazwę *pól*. Pole jest powierzchnią gładką, która odbija światło o większym natężeniu. Zmiany między wgłębieniami a polami są wykrywane przez fotoczujnik i zamieniane na sygnał cyfrowy.



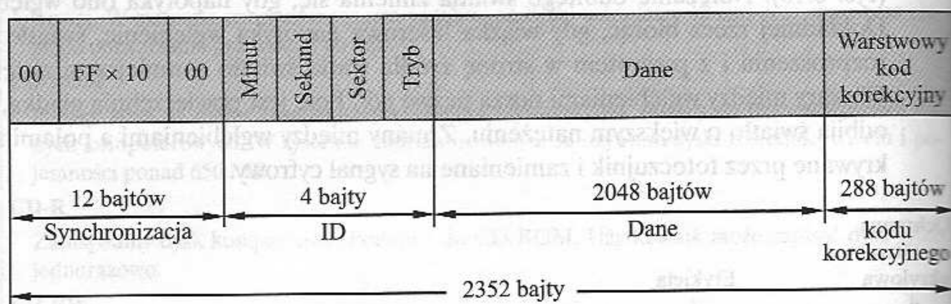
Rysunek 6.10. Działanie dysku kompaktowego

Czujnik kontroluje powierzchnię w regularnych odstępach. Początek lub koniec wgłębienia reprezentuje 1; gdy między odstępami nie ma żadnych zmian, zapisane jest 0.

Podobnie jak w przypadku dysku magnetycznego, informacje są zapisywane w postaci koncentrycznych ścieżek. W najprostszym systemie o stałej prędkości kątowej (CAV) liczba bitów na ścieżkę jest stała. Wzrost gęstości osiąga się, stosując zapis

wielostrefowy, przy którym powierzchnia jest dzielona na pewną liczbę stref, przy czym strefy dalsze od środka zawierają więcej bitów niż bliższe. Chociaż taka technika prowadzi do zwiększenia pojemności, wciąż nie jest jednak optymalna.

Aby uzyskać większą pojemność, informacje na płytach kompaktowych i CD-ROM-ach nie są organizowane w postaci koncentrycznych ścieżek. Zamiast tego dyski te zawierają pojedynczą ścieżkę spiralną, rozpoczynającą się od środka i przebiegającą spiralnie do zewnętrznej krawędzi dysku. Sektory bliskie krawędzi mają taką długość jak sektory bliskie środka. Informacje są więc rozmieszczane na dysku równomiernie, w segmentach o takich samych rozmiarach. Są one skanowane z tą samą szybkością przez obracanie dysku z prędkością zmienną. Wgłębienia są następnie odczytywane za pomocą lasera ze **stałą prędkością liniową** (*constant linear velocity* – CLV). Dysk obraca się wolniej, gdy są odczytywane dane w pobliżu krawędzi. Pojemność ścieżki i opóźnienie obrotowe w tym obszarze są więc większe. Pojemność danych CD-ROM-ów wynosi około 680 MB.



Rysunek 6.11. Format bloku CD-ROM

Dane na płycie CD-ROM są zorganizowane w postaci ciągu bloków. Typowy format bloku widać na rys. 6.11. Blok składa się z następujących pól:

- **Synchronizacja.** Pole identyfikujące początek bloku; składa się z bajta samych 0, 10 bajtów samych 1 oraz z bajta samych 0.
- **Nagłówek.** Zawiera adres bloku i bajt trybu. Tryb 0 oznacza czyste pole danych, tryb 1 – użycie kodu korekcyjnego i 2048 bajtów danych, a tryb 2 – 2336 bajtów danych użytkownika bez kodu korekcyjnego.
- **Dane.** Dane użytkownika.
- **Pomocnicze:** Dodatkowe dane użytkownika w trybie 2; w trybie 1 jest to 288-bajtowy kod korekcyjny.

W przypadku używania napędu CLV dostęp swobodny jest trudniejszy. Lokalizowanie określonego adresu wymaga wstępnego przesunięcia głowicy, dostosowania prędkości obrotowej, odczytania adresu, a następnie dokonania nieznacznych korekcji w celu znalezienia i uzyskania dostępu do potrzebnego sektora.

Dysk CD-ROM jest odpowiedni do rozpowszechniania dużych ilości danych wśród dużej liczby użytkowników. Ze względu na koszt procesu pierwotnego zapisu

nie jest odpowiedni do zastosowań jednostkowych. W porównaniu z tradycyjnymi dyskami magnetycznymi CD-ROM ma dwie zalety:

- Dysk optyczny z zapisanymi informacjami może być tanio powielany masowo – w przeciwieństwie do dysków magnetycznych. Baza danych znajdująca się na dysku twardym musi być reprodukowana dysk po dysku przy użyciu dwóch napędów.
- Dysk optyczny jest dyskiem wymiennym; umożliwia to wykorzystywanie samego dysku do archiwizacji. Większość dysków magnetycznych to dyski niewymienne. Informacja musi być skopiowana na taśmę, zanim dysk z napędem będą gotowe do wprowadzania nowej informacji.

Wady dysku CD-ROM są następujące:

- Jest to pamięć stała i nie może być aktualizowana.
- Ma czas dostępu o wiele dłuższy niż magnetyczna pamięć dyskowa, wynoszący nawet 0,5 s.

Zapisywalne dyski kompaktowe

Z myślą o zastosowaniach, w których jest potrzebna tylko niewielka liczba kopii określonego zbioru danych, opracowano dyski kompaktowe umożliwiające jednokrotny zapis i wielokrotny odczyt, znane jako *zapisywalne dyski kompaktowe* (*CD recordable* – CD-R). W tym przypadku dysk jest przygotowywany w taki sposób, aby można było następnie dokonać jednorazowego zapisu wiązką lasera o umiarkowanej intensywności. Dysponując więc nieco bardziej kosztownym sterownikiem dysku niż w przypadku CD-ROM, klient może jednorazowo zapisać dysk, następnie zaś go odczytywać.

Nośnik CD-R jest podobny do CD i CD-ROM, jednak nie jest z nimi identyczny. W przypadku dysków CD i CD-ROM informacje są zapisywane przez wytwarzanie wgłębień w powierzchni nośnika, co prowadzi do zmian współczynnika odbicia światła. W przypadku CD-R nośnik zawiera warstwę barwnika. Barwnik służy do modyfikowania współczynnika odbicia i jest aktywowany za pomocą wiązki lasera o dużym natężeniu. Powstający w ten sposób dysk może być odczytywany w napędzie CD-R lub CD-ROM.

Dyski optyczne CD-R doskonale nadają się do archiwalnego przechowywania dokumentów i plików. Umożliwiają trwałe zapisywanie wielkich ilości danych użytkownika.

Wielokrotnie zapisywalny dysk kompaktowy

Dysk optyczny CD-RW może być zapisywany wielokrotnie, podobnie jak dysk magnetyczny. Chociaż wypróbowano wiele rozwiązań, jedyne rozwiązanie czysto optyczne, które okazało się atrakcyjne, nosi nazwę **przemiany fazowej**. W dysku opartym na przemianie fazowej jest używany materiał, który w dwóch różnych stanach fazowych ma znacznie różniące się od siebie współczynniki odbicia światła. Istnieje mianowicie stan amorficzny, w którym molekuły są zorientowane przypadkowo i odbijana jest niewielka część światła; jest też stan krystaliczny o gładkiej powierzchni, odbijający

światło znacznie lepiej. Wiązka lasera jest w stanie zmieniać stan materiału z jednej fazy do drugiej. Główną wadą dysków optycznych opartych na przemianie fazowej jest to, że po pewnym czasie materiał trwale traci swoje pożądane własności. Obecnie stosowane materiały przetrwały od 500 000 do 1 000 000 cykli wymazywania.

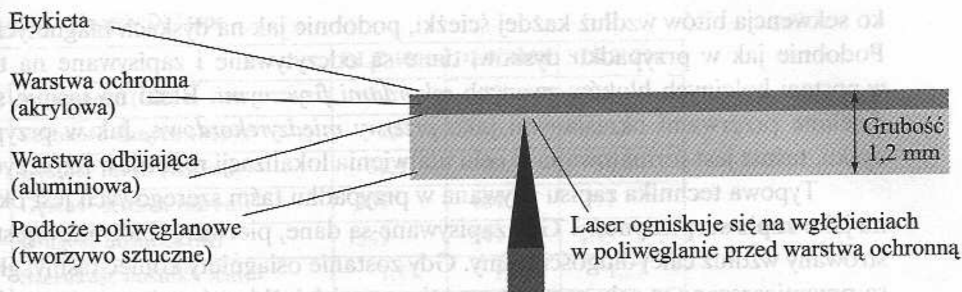
CD-RW ma oczywistą przewagę nad CD-ROM i CD-R, ponieważ można je wielokrotnie zapisywać i dzięki temu umożliwiają przechowywanie wtórne. Pod tym względem konkurują one z dyskami magnetycznymi. Podstawową zaletą dysków optycznych jest jednak to, że tolerancje ich wykonania są znacznie łagodniejsze niż w przypadku dysków magnetycznych o dużej pojemności. Z tego powodu wykazują one większą niezawodność i trwałość.

Wszechstronny dysk cyfrowy

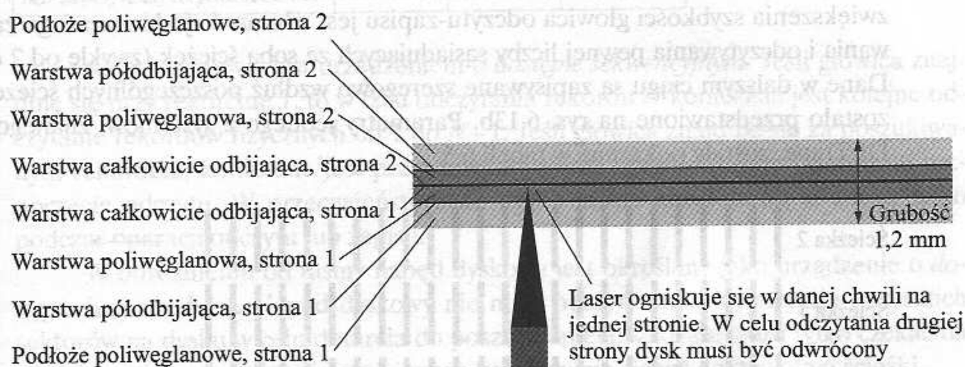
W postaci wszechstronnego dysku cyfrowego (*Digital versatile disk* – DVD) przemysł elektroniczny znalazł w końcu powszechnie uznany zamiennik analogowej taśmy wideo VHS. DVD zastąpi kasety wideo używane w magnetowidach i – co jest ważniejsze dla potrzeb naszych rozważań – zastąpi CD-ROM-y w komputerach osobistych i serwerach. DVD wprowadzi wideo w wiek cyfrowy. Jest nośnikiem filmów o wspaniałej jakości obrazu i umożliwia dostęp swobodny podobnie jak płyty akustyczne CD, które również mogą być odtwarzane w urządzeniach DVD. Na dysku mogą być umieszczane ogromne ilości danych; obecnie jest to siedem razy tyle co na CD-ROM. Dzięki ogromnej pojemności DVD i wysokiej jakości obrazu, gry komputerowe staną się bardziej realistyczne, a oprogramowanie edukacyjne będzie zawierało więcej nagrań wideo. Następstwem tych dokonań będzie kolejne nasilenie ruchu w Internecie i w sieciach firmowych, gdy te materiały zostaną wcielone do witryn WWW. Większa pojemność DVD wynika z trzech różnic w stosunku do dysków kompaktowych (rys. 6.12):

1. W DVD bity są upakowane gęściej. Odstęp między zwojami spirali w płytach kompaktowych wynosi 1,6 μm, a minimalna odległość między wgłębieniami wzdłuż spirali jest równa 0,834 μm. W technice DVD jest używany laser o mniejszej długości fali, dzięki czemu odstęp zwojów wynosi 0,74 μm, minimalny zaś odstęp między wgłębieniami – 0,4 μm. Wynikiem tych udoskonaleń jest mniej więcej siedmiokrotny wzrost pojemności, do poziomu około 4,7 GB.
2. W DVD wykorzystuje się drugą warstwę wgłębień i pól leżącą na pierwszej warstwie. Dwuwarstwowy DVD ma częściowo odbijającą warstwę na warstwie odbijającej i przez dostosowanie ogniska lasery w napędach DVD są w stanie odczytywać każdą warstwę oddzielnie. Technika ta niemal podwaja pojemność dysku, do poziomu około 8,5 GB. Niepełne podwojenie wynika z tego, że pojemność drugiej warstwy jest ograniczona z powodu słabszego odbijania światła.
3. DVD-ROM mogą być dwustronne, podczas gdy na płycie kompaktowej dane są zapisywane tylko na jednej stronie. Dzięki temu łączna pojemność może sięgać 17 GB.

Podobnie jak w przypadku CD, dyski DVD są również wytwarzane w wersjach zapisywalnych (tabela 6.4).



(a) CD-ROM – pojemność 628 MB



(b) DVD-ROM, obustronny, dwuwarstwowy – pojemność 17 GB

Rysunek 6.12. CD-ROM i DVD-ROM

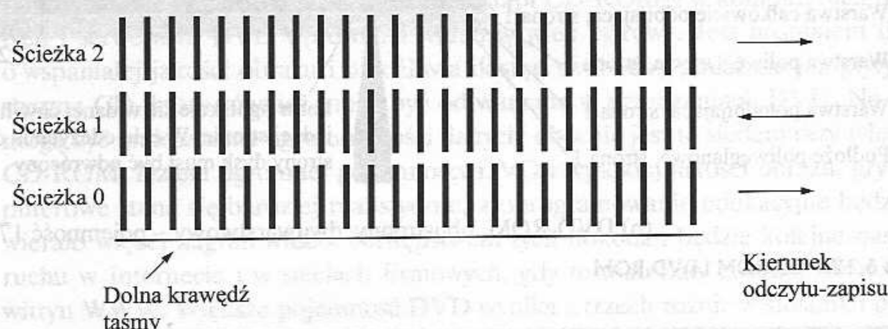
6.4. Taśma magnetyczna

W systemach taśmowych wykorzystuje się te same metody odczytu i zapisu co w systemach dyskowych. Nośnikiem jest elastyczna taśma poliestrowa pokryta materiałem magnetyzowanym. Pokrycie to może się składać z cząsteczek czystego metalu w specjalnym spoiwie lub może nim być naparowana warstwa metalowa. Taśma i napęd taśmy są podobne do wykorzystywanych w domowych systemach magnetofonowych. Szerokości taśm sięgają od 0,38 cm do 1,27 cm. Taśmy były zwykle dostarczane w otwartych szpulach i przed użyciem wymagały przewlekania przez drugą szpulę. Obecnie praktycznie wszystkie taśmy są dostarczane w kasetach.

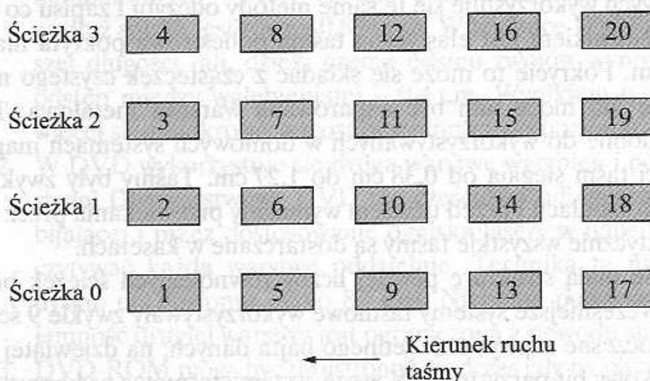
Dane na taśmie mają strukturę pewnej liczby równoległych ścieżek biegnących wzdłuż taśmy. Wcześniejsze systemy taśmowe wykorzystywały zwykle 9 ścieżek. Umożliwiało to jednoczesne zapisywanie jednego bajta danych; na dziewiątej ścieżce zapisywano dodatkowy bit parzystości. Nowsze systemy taśmowe wykorzystują 18 lub 36 ścieżek, co odpowiada słowu lub podwójnemu słowu. Rejestrowanie danych w tej postaci jest określane jako **zapis równoległy**. Zamiast tego w większości nowoczesnych systemów jest używane **zapis szeregowy**, przy którym dane są układane ja-

ko sekwencja bitów wzdłuż każdej ścieżki, podobnie jak na dyskach magnetycznych. Podobnie jak w przypadku dysków, dane są odczytywane i zapisywane na taśmie w postaci kolejnych bloków zwanych *rekordami fizycznymi*. Bloki na taśmie są oddzielane przerwami określanymi jako *przerwy międzyrekordowe*. Jak w przypadku dysku, taśma jest formatowana w celu ułatwienia lokalizacji rekordów fizycznych.

Typowa technika zapisu używana w przypadku taśm szeregowych jest określana jako **zapis serpentynowy**. Gdy zapisywane są dane, pierwszy zbiór bitów jest rejestrowany wzdłuż całej długości taśmy. Gdy zostanie osiągnięty koniec taśmy, głowice są przemieszczane w celu zarejestrowania nowej ścieżki; taśma jest znów zapisywana na całej długości, tym razem w kierunku przeciwnym. Proces ten jest kontynuowany do przodu i do tyłu, dopóki taśma nie zostanie zapełniona (rys. 6.13a). W celu zwiększenia szybkości głowica odczytu-zapisu jest zdolna do jednoczesnego zapisywania i odczytywania pewnej liczby sąsiadujących ze sobą ścieżek (zwykle od 2 do 8). Dane w dalszym ciągu są zapisywane szeregowo wzdłuż poszczególnych ścieżek, co zostało przedstawione na rys. 6.13b. Parametry jednego z systemów, znanego jako DLTtape, zostały pokazane w tabeli 6.5.



(a) Odczyt i zapis serpentynowy



(b) Układ blokowy systemu jednoczesnego odczytu-zapisu czterech ścieżek

Rysunek 6.13. Typowe własności taśmy magnetycznej

Tabela 6.5. Napędy DLTtape

	DLT 4000	DLT 8000	SDLT 220
Pojemność (GB)	20	40	110
Szybkość danych (MB/s)	1,5	6,0	11,0
Gęstość bitów (KB/cm)	32,3	38,6	51,6
Gęstość ścieżek (na cm)	101	164	317
Długość nośnika (m)	549	549	549
Szerokość nośnika (cm)	1,27	1,27	1,27
Liczba ścieżek	128	208	448
Liczba ścieżek odczytywanych lub zapisywanych jednocześnie	2	4	8

Napęd taśmowy jest urządzeniem o *dostępie sekwencyjnym*. Jeśli głowica znajduje się przy rekordzie 1, to w celu odczytania rekordu N konieczne jest kolejne odczytanie rekordów fizycznych od 1 do $N-1$. Jeśli głowica znajduje się za poszukiwanym rekordem, konieczne jest przewinięcie pewnej długości taśmy i ponowne rozpoczęcie odczytu. W przeciwieństwie do dysku taśma znajduje się w ruchu tylko podczas operacji odczytu lub zapisu.

W odróżnieniu od taśmy napęd dyskowy jest określany jako urządzenie o *dostępie bezpośrednim*. Napęd dyskowy nie musi odczytywać sekwencyjnie wszystkich sektorów na dysku w celu dotarcia do poszukiwanego sektora. Musi tylko czekać na właściwy sektor na jednej ścieżce i może sukcesywnie sięgać do dowolnej ścieżki.

Pamięć taśmowa była pierwszym rodzajem pamięci pomocniczej. Wciąż jest szeroko używana jako najtańszy i najpowolniejszy element hierarchii pamięci.

6.5. Polecana literatura i witryny WWW

Dobry przegląd techniki rejestracji będącej podstawą systemów dyskowych i taśmowych znajduje się w [MEE96a]. [MEE96b] skupia się na technikach zapisu danych w tych systemach. Krótki, lecz pouczający artykuł [COME00] jest poświęcony aktualnym tendencjom techniki zapisu na dyskach magnetycznych.

Doskonałym przeglądem techniki RAID, napisanym przez wynalazców tej koncepcji, jest [CHEN94]. Bardziej szczegółowa dyskusja została opublikowana przez RAID Advisory Board, będące stowarzyszeniem dostawców i użytkowników wyrobów związanych z RAID [MASS97]. Interesującym artykułem jest [FRIE96].

Praca [MARC90] zawiera doskonały przegląd pamięci optycznych. Dobrym przeglądem techniki zapisu i odczytu w tych pamięciach jest [MANS97].

Obszerny opis wszystkich typów zewnętrznych systemów pamięci znajduje się w [ROSC99], przy czym ilość szczegółów technicznych na temat każdego z nich jest umiarkowana. Innym dobrym przeglądem jest [KHUR01].

CHEN94 Chen P., Lee E., Gibson G., Katz R., Patterson D.: „RAID: High-Performance, Reliable Secondary Storage". *ACM Computing Surveys*, June 1994.

COME00 Comerford R.: „Magnetic storage; The Medium that Wouldn't Die". *IEEE Spectrum,* December 2000.

FRIE96 Friedman M.: „RAID keeps Going and Going and…" *IEEE Spectrum,* April 1996.

KHUR01 Khurshudov A.: *The Essential Guide to Computer Data Storage*. Upper Saddle River, Prentice Hall, 2001.

MANS97 Mansuripur M., Sincerbox G.: „Principles and Techniques of Optical Data Storage". *Proceedings of the IEEE,* November 1997.

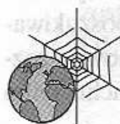
MARC90 Marchant A.: *Optical Recording.* Reading, MA: Addison-Wesley, 1990.

MASS97 Massiglia P.: *The RAID book: A Storage System Technology Handbook.* St. Peter, The Raid Advisory Board, 1997.

MEE96a Mee C., Daniel E. (eds.): *Magnetic Recording Technology.* New York, McGraw-Hill, 1996.

MEE96b Mee C., Daniel E. (eds.): *Magnetic Storage Handbook*. New York, McGraw-Hill, 1996.

ROSC99 Rosch W.: *Winn L. Rosch Hardware Bible*. Indianapolis, Sams, 1999.


Polecane witryny WWW:

- **RAID Advisory Group.** Grupa przemysłowa RAID. Informacje o technologii i produktach RAID.
- **Optical Storage Technology Association.** Dobre źródło informacji o technologii pamięci optycznych i o wytwórcach, a także obszerny wykaz odpowiednich łącz.
- **DLTtape.** Dobry zbiór informacji technicznych i łącz do wytwórców.
- **Data Storage Magazine.** Witryna WWW tego czasopisma zawiera obszerne informacje o produktach służących do przechowywania danych i o ich wytwórcach.

6.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

CD – płyta kompaktowa – *compact disk*
CD-R – zapisywalny dysk kompaktowy – *compact disk recordable*
CD-ROM – pamięć stała na dysku kompaktowym – *compact disk read-only-memory*
CD-RW – wielokrotnie zapisywalny dysk kompaktowy – *compact disk rewritable*
Cylinder – *cylinder*
Czas dostępu – *access time*
Czas przeszukiwania – *seek time*
Czas transferu – *transfer time*
Dane paskowane – *striped data*
DVD – cyfrowy dysk wideo – *digital video disk*
DVD-R – zapisywalny DVD – *DVD recordable*
DVD-ROM – pamięć stała na dysku DVD – *DVD read-only memory*
DVD-RW – wielokrotnie zapisywalny dysk DVD – *DVD rewritable*
Dysk magnetyczny – *magnetic disk*
Dysk niewymienny – *nonremovable disk*
Dysk wymienny – *removable disk*
Dysk z głowicą nieruchomą – *fixed-head disk*
Dysk z głowicą ruchomą – *movable-head disk*
Dyskietka – *floppy disk*
Głowica – *head*
Magnetooporowy – *magnetoresistive*

Opóźnienie obrotowe – *rotational latency*
Pamięć optyczna – *optical memory*
Płyta – *platter*
Podłoże – *substrate*
Pole – *land*
Przerwa – *gap*
RAID – redundancyjna tablica niezależnych dysków – *redundant array of independent disks*
Sektor – *sector*
Stała prędkość kątowa (CAV) – *constant angular velocity*
Stała prędkość liniowa (CLV) – *constant linear velocity*
Ścieżka – *track*
Taśma magnetyczna – *magnetic tape*
Wgłębienie – *pit*
Zapis serpentynowy – *serpentine recording*
Zapis wielostrefowy – *multiple zoned recording*

Pytania kontrolne

6.1. Jakie korzyści wynikają z używania podłoża szklanego w dyskach magnetycznych?
6.2. W jaki sposób zapisuje się dane na dysku magnetycznym?
6.3. W jaki sposób odczytuje się dane z dysku magnetycznego?
6.4. Objaśnij różnicę między prostym systemem CAV a systemem zapisu wielostrefowego.
6.5. Zdefiniuj wyrażenia *ścieżka*, *cylinder* i *sektor*.
6.6. Jaki jest typowy rozmiar sektora dysku?
6.7. Zdefiniuj wyrażenia *czas przeszukiwania*, *opóźnienie obrotowe*, *czas dostępu* i *czas transferu*.
6.8. Jakie są wspólne własności wszystkich poziomów RAID?
6.9. Krótko zdefiniuj siedem poziomów RAID.
6.10. Objaśnij wyrażenie *dane paskowane*.
6.11. W jaki sposób uzyskuje się redundancję w systemie RAID?
6.12. Jakie jest rozróżnienie między dostępem równoległym a bezpośrednim w kontekście RAID?
6.13. Jaka jest różnica między CAV i CLV?
6.14. Jakie różnice między CD a DVD sprawiają, że to ostatnie rozwiązanie charakteryzuje się większą pojemnością?
6.15. Objaśnij zapis serpentynowy.

Problemy do rozwiązania

6.1. Rozważ dysk o N ścieżkach ponumerowanych od 0 do $(N - 1)$ i załóż, że sektory, na które zgłoszono zapotrzebowanie, są przypadkowo i równomiernie rozłożone na dysku. Zamierzamy obliczyć średnią liczbę ścieżek przekraczaną podczas przeszukiwania.

(a) Najpierw oblicz prawdopodobieństwo przeszukania długości j, gdy głowica znajduje się aktualnie nad ścieżką t. *Wskazówka*: jest to problem wyznaczenia całkowitej liczby kombinacji przy uwzględnieniu, że w przeszukiwaniu wszystkie miejsca ścieżek docelowych są równie prawdopodobne.

(b) Oblicz następnie prawdopodobieństwo przeszukania długości K. *Wskazówka*: obejmuje to podsumowanie wszystkich możliwych kombinacji ruchów K ścieżek.

(c) Oblicz średnią liczbę ścieżek przekraczanych przy przeszukiwaniu, posługując się wzorem na wartość oczekiwaną:

$$E[x] = \sum_{i=0}^{N-1} i \times \Pr[x = i]$$

Wskazówka: użyj równości

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}; \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

(d) Wykaż, że dla dużych N średnia liczba ścieżek przekraczanych podczas przeszukiwania osiąga $N/3$.

6.2. Zdefiniuj następujące parametry systemu dyskowego:

t_s – czas przeszukiwania; średni czas umieszczania głowicy nad ścieżką,
r – prędkość obrotowa dysku, w obrotach na sekundę,
n – liczba bitów na sektor,
N – pojemność ścieżki, w bitach,
t_A – czas dostępu do sektora.

Wyprowadź wzór na t_A jako funkcję pozostałych parametrów.

6.3. Przyjmij 10-napędową konfigurację RAID. Wypełnij następującą tablicę, służącą do porównania różnych poziomów RAID:

Poziom RAID	Gęstość upakowania	Szerokość pasma	Wydajność transakcyjna
0	1		1
1			
2			
3		1	
4			
5			

Każdy parametr jest znormalizowany w stosunku do tego poziomu RAID, który zapewnia najlepszą wydajność; pozostałe liczby w tablicy powinny mieć wartości pomiędzy 0 a 1. Gęstość upakowania pamięci określa część pojemności dysków dostępną dla danych użytkownika. Szerokość pasma odzwierciedla szybkość transferu danych z tablicy. Wydajność transakcyjna jest miarą liczby operacji wejścia-wyjścia na sekundę, którą może realizować tablica.

6.4. Powinno być oczywiste, że paskowanie dysku może poprawić szybkość przesyłania danych, jeśli rozmiar paska jest mały w porównaniu z rozmiarem żądania wejścia-wyjścia. Powinno być również oczywiste, że RAID 0 umożliwia zwiększenie wydajności w stosunku do pojedynczego, dużego dysku, ponieważ wiele żądań wejścia-wyjścia może być obsługiwanych równolegle. Jednakże, czy w tym ostatnim przypadku paskowanie dysku jest konieczne? To znaczy, czy paskowanie dysku umożliwia zwiększenie szybkości odpowiedzi na żądania wejścia-wyjścia w porównaniu z analogiczną tablicą dysków bez paskowania?

Rozdział 7

Wejście-wyjście

7.1. Urządzenia zewnętrzne
Klawiatura i monitor
Napęd dyskowy
7.2. Moduły wejścia-wyjścia
Działanie modułu
Struktura modułu wejścia-wyjścia
7.3. Programowane wejście-wyjście
Przegląd
Rozkazy wejścia-wyjścia
Instrukcje wejścia-wyjścia
7.4. Wejście-wyjście sterowane przerwaniami
Przetwarzanie przerwań
Zagadnienia projektowania
Sterownik przerwań Intel 82C59A
Programowalny interfejs urządzeń peryferyjnych Intel 82C55A
7.5. Bezpośredni dostęp do pamięci
Wady wejścia-wyjścia programowanego i sterowanego przerwaniami
Działanie DMA
7.6. Kanały i procesory wejścia-wyjścia
Ewolucja funkcjonowania wejścia-wyjścia
Właściwości kanałów wejścia-wyjścia
7.7. Interfejs zewnętrzny: FireWire i InfiniBand
Rodzaje interfejsów
Konfiguracje dwupunktowe i wielopunktowe
Magistrala szeregowa FireWire
InfiniBand
7.8. Polecana literatura i witryny WWW
7.9. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

PODSTAWOWE SPOSTRZEŻENIA

- Architektura wejścia-wyjścia systemu komputerowego jest jego interfejsem ze światem zewnętrznym. Architektura ta jest projektowana tak, aby zapewnić systematyczne środki kontrolowanego współdziałania ze światem zewnętrznym i aby dostarczyć systemowi operacyjnemu informacji wymaganych do skutecznego zarządzania działaniami wejścia-wyjścia.
- Istnieją trzy główne techniki wejścia-wyjścia: **programowane wejście-wyjście**, w którym operacje wejścia-wyjścia są prowadzone pod bezpośrednią i ciągłą kontrolą programu zgłaszającego zapotrzebowanie na takie operacje; **wejście-wyjście sterowane przerwaniami**, w którym program wydaje polecenie wejścia-wyjścia, po czym kontynuuje pracę do czasu, gdy zostanie to przerwane przez sprzęt wejścia-wyjścia sygnalizujący zakończenie operacji wejścia-wyjścia; wreszcie **bezpośredni dostęp do pamięci** (*direct memory access* – **DMA**), w którym wyspecjalizowany procesor wejścia-wyjścia przejmuje kontrolę nad operacjami wejścia-wyjścia w celu przeniesienia wielkiego bloku danych.
- Dwoma ważnymi przykładami zewnętrznych interfejsów wejścia-wyjścia są **FireWire** i **InfiniBand**.

Obok procesora i zespołu modułów pamięci trzecim kluczowym elementem systemu komputerowego jest zespół modułów wejścia-wyjścia. Każdy moduł jest dołączony do magistrali systemowej lub centralnego przełącznika i steruje jednym lub wieloma urządzeniami peryferyjnymi. Moduł wejścia-wyjścia nie jest po prostu złączem mechanicznym służącym do połączenia urządzenia z magistralą systemową. Zawiera on pewną „inteligencję", to znaczy układy logiczne umożliwiające komunikację między urządzeniem peryferyjnym a magistralą.

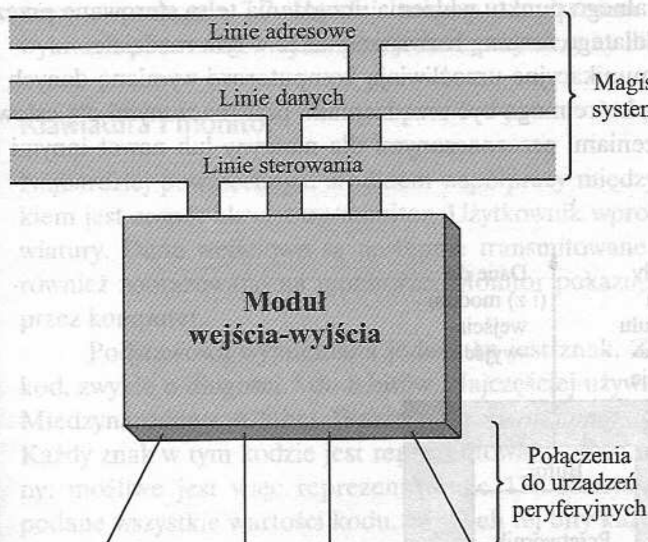
Czytelnik może się dziwić, dlaczego nie łączy się urządzeń peryferyjnych bezpośrednio z magistralą systemową. Przyczyny są następujące:

- Istnieje znaczna różnorodność urządzeń peryferyjnych, różniących się sposobem pracy. Byłoby niepraktyczne wbudowywanie niezbędnych układów logicznych do procesora w celu umożliwienia sterowania tak szerokim zakresem urządzeń.
- Szybkość przesyłania (transferu) danych do/z urządzeń peryferyjnych jest o wiele mniejsza niż w przypadku pamięci lub procesora. Jest więc niepraktyczne wykorzystywanie szybkiej magistrali systemowej do bezpośredniego komunikowania się z urządzeniami peryferyjnymi.
- Z drugiej strony, szybkość przesyłania danych niektórych urządzeń peryferyjnych jest większa niż szybkość pamięci lub procesora. Jak poprzednio – niedopasowanie mogłoby prowadzić do niewydolności systemu, gdyby nie zostały podjęte odpowiednie środki.
- Urządzenia peryferyjne wykorzystują często inne formaty danych i długości słowa niż komputery, do których są przyłączone.

Potrzebny jest więc moduł wejścia-wyjścia. Moduł ten gra dwie role (rys. 7.1):

- Interfejsu z procesorem i z pamięcią poprzez magistralę systemową lub centralnego przełącznika.
- Interfejsu z jednym lub wieloma urządzeniami peryferyjnymi przez dostosowane łącza danych.

Rozpoczniemy ten rozdział od krótkiego przeglądu urządzeń zewnętrznych, po czym omówimy strukturę i funkcję modułu wejścia-wyjścia. Rozpatrzymy następnie różne sposoby realizowania funkcji modułu wejścia-wyjścia we współpracy z procesorem i z pamięcią, tzn. wewnętrzny interfejs wejścia-wyjścia. Na zakończenie przeanalizujemy zewnętrzny interfejs wejścia-wyjścia między modułem wejścia-wyjścia a światem zewnętrznym.



Rysunek 7.1. Ogólny model modułu wejścia-wyjścia

7.1. Urządzenia zewnętrzne

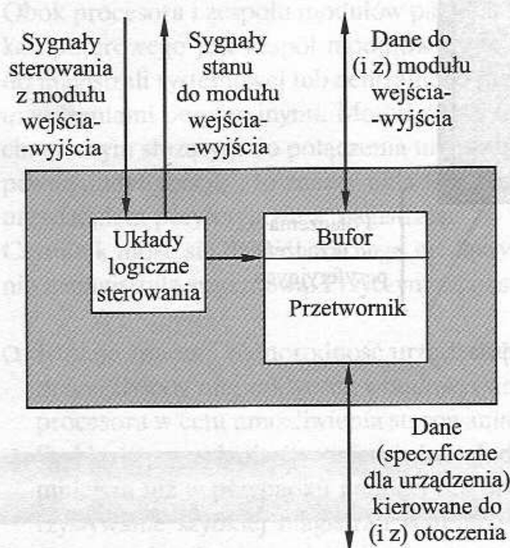
Operacje wejścia-wyjścia są realizowane za pomocą szerokiego asortymentu urządzeń zewnętrznych, które umożliwiają wymianę danych między otoczeniem zewnętrznym a komputerem. Urządzenie zewnętrzne współpracuje z komputerem poprzez łącze z modułem wejścia-wyjścia (rys. 7.1). Łącze jest używane do wymiany sygnałów sterowania i stanu oraz danych między modułem wejścia-wyjścia a urządzeniem zewnętrznym. Urządzenie zewnętrzne połączone z modułem wejścia-wyjścia jest często określane jako **urządzenie peryferyjne**.

Możemy podzielić urządzenia zewnętrzne na trzy kategorie:

- **Przeznaczone do odczytywania przez człowieka.** Odpowiednie do komunikowania się z użytkownikiem komputera.
- **Przeznaczone do odczytywania przez maszynę.** Odpowiednie do komunikowania się ze sprzętem.
- **Komunikacyjne.** Odpowiednie do komunikowania się z odległymi urządzeniami.

Przykładami urządzeń przeznaczonych dla człowieka są terminale wizyjne (*video display terminal* – VDT) i drukarki. Przykładami urządzeń przeznaczonych dla maszyny są dyski magnetyczne, systemy taśmowe oraz czujniki i urządzenia wykonawcze wykorzystywane w robotach. Zauważmy, że w tym rozdziale traktujemy systemy dyskowe i taśmowe jako urządzenia wejścia-wyjścia, podczas gdy w rozdz. 6 uważaliśmy je za urządzenia pamięci. Z funkcjonalnego punktu widzenia urządzenia te stanowią część hierarchii pamięci i dlatego mówienie o nich w rozdz. 6 było właściwe. Ze strukturalnego punktu widzenia urządzenia te są sterowane przez moduły wejścia-wyjścia i dlatego zostaną rozważone także w tym rozdziale.

Urządzenia komunikacyjne umożliwiają komputerowi wymianę danych z odległymi urządzeniami, które mogą być urządzeniami przeznaczonymi dla człowieka, jak terminale, urządzeniami przeznaczonymi dla maszyny lub nawet innymi komputerami.



Rysunek 7.2. Urządzenie zewnętrzne

Natura urządzenia zewnętrznego jest pokazana na rys. 7.2 w bardzo ogólnym ujęciu. Interfejs z modułem wejścia-wyjścia ma postać sygnałów sterowania, stanu i danych. *Sygnały sterujące* określają funkcję, jaką ma pełnić urządzenie, np. wysłać dane do modułu wejścia-wyjścia (WEJŚCIE lub ODCZYT), przyjąć dane z modułu

wejścia-wyjścia (WYJŚCIE lub ZAPIS), poinformować o stanie lub wykonać pewne funkcje sterujące właściwe dla danego urządzenia (np. ustawić głowicę dysku). *Dane* mają postać zespołu bitów przeznaczonych do wysłania lub odbieranych z modułu wejścia-wyjścia. *Sygnały stanu* wskazują stan urządzenia. Przykładami są GOTOWOŚĆ/BRAK GOTOWOŚCI (READY/NOT READY), wskazujące, czy urządzenie jest gotowe do przesyłania danych.

Logiczne układy sterowania związane z urządzeniem sterują jego pracą w odpowiedzi na polecenia płynące z modułu wejścia-wyjścia. *Przetwornik* dokonuje konwersji danych z postaci elektrycznej na inną formę energii w przypadku wyjścia, z innych zaś form na elektryczną podczas wejścia. Zwykle z przetwornikiem jest związany bufor umożliwiający czasowe przechowywanie danych przenoszonych między modułem wejścia-wyjścia a otoczeniem zewnętrznym; bufor ten ma zwykle rozmiar 8÷16 bitów.

Interfejs między modułem wejścia-wyjścia a urządzeniem zewnętrznym omówimy w podrozdz. 7.7. Interfejs między urządzeniem zewnętrznym a otoczeniem wykracza poza zakres tej książki, jednak przytoczymy kilka krótkich przykładów.

Klawiatura i monitor

Najbardziej powszechnym środkiem współpracy między komputerem a użytkownikiem jest zespół klawiatura/monitor. Użytkownik wprowadza dane za pomocą klawiatury. Dane wejściowe są następnie transmitowane do komputera i mogą być również zobrazowane na monitorze. Monitor pokazuje również dane dostarczane przez komputer.

Podstawową wymienianą jednostką jest znak. Z każdym znakiem wiąże się kod, zwykle o długości 7 do 8 bitów. Najczęściej używanym kodem tekstowym jest Międzynarodowy Alfabet Wzorcowy (*International Reference Alphabet* – IRA)[1]. Każdy znak w tym kodzie jest reprezentowany przez unikalny 7-bitowy kod binarny; możliwe jest więc reprezentowanie 128 różnych znaków[2]. W tabeli 7.1 są podane wszystkie wartości kodu. W tabeli tej bity każdego znaku są oznaczane od b_7, który jest najbardziej znaczącym bitem, do b_1 – najmniej znaczącego bitu. Występują dwa typy znaków: drukowalne i kontrolne (tabela 7.2). Drukowalnymi znakami są znaki alfabetu, cyfry i znaki specjalne, które mogą być drukowane na papierze lub wyświetlane na ekranie. Bitową reprezentacją na przykład znaku „K" jest $b_7b_6b_5b_4b_3b_2b_1 = 1001011$. Niektóre ze znaków kontrolnych są wykorzystywane przy drukowaniu lub wyświetlaniu znaków; przykładem jest znak powrotu karetki. Inne znaki kontrolne są związane z procedurami komunikacyjnymi.

[1] Kod IRA zdefiniowany w dokumencie ITU-T Recommendation T.50 był uprzednio znany jako International Alphabet Numer 5 (IA5). Przyjęta w USA wersja narodowa IRA jest znana jako American Standard Code for Information Inchange (ASCII).

[2] Znaki kodowane za pomocą IRA są prawie zawsze przechowywane lub przesyłane po 8 bitów na znak. Ósmy bit jest bitem parzystości wykorzystywanym do wykrywania błędów. Bit parzystości znajduje się w pozycji najbardziej znaczącego bitu i dlatego jest symbolizowany jako b_8.

Tabela 7.1. Międzynarodowy Alfabet Wzorcowy (IRA)

	b_7			0	0	0	0	1	1	1	1
		b_6		0	0	1	1	0	0	1	1
			b_5	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1								
0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	”	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	'	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	_	o	DEL

Tabela 7.2. Znaki sterujące IRA

Kontrola formatu	
BS (cofnięcie; *backspace*): oznacza przesunięcie mechanizmu drukującego lub kursora wstecz o jedną pozycję.	**T** (tabulacja pionowa; *vertical tab*): oznacza przesunięcie mechanizmu drukującego lub kursora do następnego z szeregu z góry określonych wierszy.
HT (tabulacja pozioma; *horizontal tab*): oznacza przesunięcie mechanizmu drukującego lub kursora do następnej z góry określonej pozycji tabulacyjnej lub do pozycji zatrzymania.	**FF** (wysunięcie strony; *form feed*): oznacza przesunięcie mechanizmu drukującego lub kursora do początkowej pozycji na następnej stronie lub na ekranie.
LF (przesunięcie o wiersz; *line feed*): oznacza przesunięcie mechanizmu drukującego lub kursora do początku następnego wiersza.	**CR** (powrót karetki; *carriage return*): oznacza przesunięcie mechanizmu drukującego lub kursora do początku tego samego wiersza.
Kontrola transmisji	
SOH (początek nagłówka; *start of heading*): wskazuje początek nagłówka, który może zawierać adres lub informacje o trasie.	**ACK** (potwierdzenie odbioru; *acknowledge*): znak transmitowany przez przyrząd odbierający jako odpowiedź potwierdzająca skierowana do nadawcy; używany jako odpowiedź pozytywna na komunikaty odpytywania.
STX (początek tekstu; *start of text*): wskazuje początek tekstu, a także koniec nagłówka.	

Tabela 7.2. Znaki sterujące IRA (cd.)

ETX (koniec tekstu; *end of text*): używany do zakończenia tekstu rozpoczętego za pomocą STX. **EOT** (koniec transmisji; *end of transmission*): oznacza koniec transmisji jednego lub więcej tekstów opatrzonych nagłówkiem. **ENQ** (zapytanie; *enquiry*): zapytanie o odpowiedź odległej stacji; może być używane jako zapytanie „Kim jesteś?" kierowane w celu identyfikacji stacji.	**NAK** (potwierdzenie negatywne; *negative acknowledgment*): znak transmitowany przez przyrząd odbierający jako odpowiedź negatywna skierowana do nadawcy; używany jako odpowiedź negatywna na komunikaty odpytywania. **SYN** (synchronizacja; *synchronous/idle*): używany do osiągnięcia synchronizacji w synchronicznych systemach transmisyjnych; gdy nie są wysyłane dane, system taki może ciągle nadawać znaki SYN. **ETB** (koniec bloku transmisji; *end of transmission block*): wskazuje koniec bloku danych przy połączeniu; używany do formowania bloków danych, gdy struktura blokowa niekoniecznie wynika z formatu przetwarzania.
Separatory informacji	
FS (separator plików; *file separator*) **GS** (separator grup; *group separator*) **RS** (separator rekordów; *record separator*) **US** (separator połączony; *united separator*)	Separatory informacji mogą być używane dowolnie, jednak z zachowaniem hierarchii od FS (najbardziej dopuszczającej) do US (najmniej dopuszczającej).
Różne	
NUL (brak znaku; *null*): używany do wypełniania czasu lub wypełniania przestrzeni na taśmie, gdy nie ma danych. **BEL** (znak wywołujący sygnał akustyczny; *bell*): używany, gdy istnieje potrzeba zwrócenia uwagi obsługi; może sterować alarmem lub przyrządami zwracającymi uwagę. **SO** (przełączenie na zestaw niestandardowy; *shift out*): wskazuje, że następne kombinacje kodowe powinny być interpretowane jako niezgodne z zestawem standardowym, aż do wystąpienia znaku SI. **SI** (przełączenie na zestaw standardowy; *shift in*): wskazuje, że następne kombinacje kodowe powinny być interpretowane zgodnie z zestawem standardowym. **DEL** (usuwanie; *delete*): używany do usuwania niepożądanych znaków; na przykład przez nadpisywanie. **SP** (odstęp; *space*): znak niedrukowany, służący do oddzielania słów lub do przesuwania mechanizmu drukującego albo kursora o jedną pozycję do przodu.	**DLE** (znak sterujący transmisją; *data link escape*): znak zmieniający znaczenie jednego lub wielu następnych znaków. Umożliwia dodatkowe sterowanie lub pozwala na wysyłanie znaków danych o dowolnej kombinacji bitów. **DC1, DC2, DC3 i DC4** (znaki sterowania przyrządami; *device controls*): znaki sterowania przyrządami pomocniczymi lub specjalnymi właściwościami terminali. **CAN** (kasowanie; *cancel*): wskazuje, że dane poprzedzające ten znak w komunikacie lub w bloku powinny być zaniedbane (zwykle z powodu wykrycia błędu). **EM** (fizyczne zakończenie nośnika; *end of medium*): wskazuje na fizyczne zakończenie karty, taśmy lub innego nośnika czy też zakończenie wymaganej lub używanej części nośnika. **SUB** (zastąpienie; *substitute*): wstawiany zamiast znaku, który okazał się błędny lub nieważny. **ESC** (znak unikowy; *escape*): znak umożliwiający rozszerzenie kodu w ten sposób, że nadaje określonej liczbie następnych znaków zmienione znaczenie.

W przypadku wejścia z klawiatury naciśnięcie klawisza przez użytkownika powoduje wygenerowanie sygnału elektrycznego, który jest interpretowany przez przetwornik w klawiaturze i tłumaczony na wzór bitowy odpowiedniego kodu IRA. Ten wzór bitowy jest następnie transmitowany do modułu wejścia-wyjścia w komputerze. W samym komputerze tekst może być przechowywany w tym samym kodzie IRA. W przypadku wyjścia kody znaków IRA są transmitowane z modułu wejścia-wyjścia do urządzenia zewnętrznego. Przetwornik w tym urządzeniu interpretuje kod i wysyła niezbędne sygnały elektryczne do urządzenia wyjściowego w celu albo wyświetlenia wskazanego znaku, albo wykonania wymaganej funkcji sterującej.

Napęd dyskowy

Napęd dysku zawiera układy elektroniczne służące do wymiany danych oraz sygnałów sterowania i stanu z modułem wejścia-wyjścia, a także układy elektroniczne sterowania mechanizmem odczytu i zapisu. W przypadku dysku z głowicą nieruchomą przetwornik może wykonać konwersję między wzorami magnetycznymi występującymi na poruszającej się powierzchni dysku a bitami w buforze urządzenia (rys. 7.2). Napęd dysku z ruchomą głowicą musi również być w stanie sterować radialnym przesuwaniem ramienia nad powierzchnią dysku.

7.2. Moduły wejścia-wyjścia

Działanie modułu

Główne funkcje modułu wejścia-wyjścia można podzielić na następujące kategorie:

- sterowanie i taktowanie,
- komunikacja z procesorem,
- komunikacja z urządzeniem,
- buforowanie danych,
- wykrywanie błędów.

W dowolnym przedziale czasu procesor może komunikować się z jednym lub wieloma urządzeniami zewnętrznymi według nieprzewidywalnego schematu, zależnie od potrzeb programu w odniesieniu do wejścia lub wyjścia. Zasoby wewnętrzne, takie jak pamięć główna i magistrala systemowa, mogą być wykorzystywane wspólnie przez różne operacje, w tym operacje wejścia-wyjścia danych. Dlatego właśnie do funkcji wejścia-wyjścia należy **sterowanie i taktowanie** mające na celu koordynowanie przepływu informacji między zasobami wewnętrznymi a urządzeniami zewnętrznymi. Na przykład sterowanie transferem danych z urządzenia zewnętrznego do procesora może być określone następującą sekwencją kroków:

1. Procesor żąda od modułu wejścia-wyjścia sprawdzenia stanu dołączonego urządzenia.
2. Moduł wejścia-wyjścia udziela odpowiedzi o stanie urządzenia.

3. Jeśli urządzenie działa i jest gotowe do transmitowania, procesor zgłasza zapotrzebowanie na przesłanie danych, posługując się rozkazem do modułu wejścia--wyjścia.
4. Moduł wejścia-wyjścia otrzymuje jednostkę danych (np. 8 lub 16 bitów) z urządzenia zewnętrznego.
5. Dane są przenoszone z modułu wejścia-wyjścia do procesora.

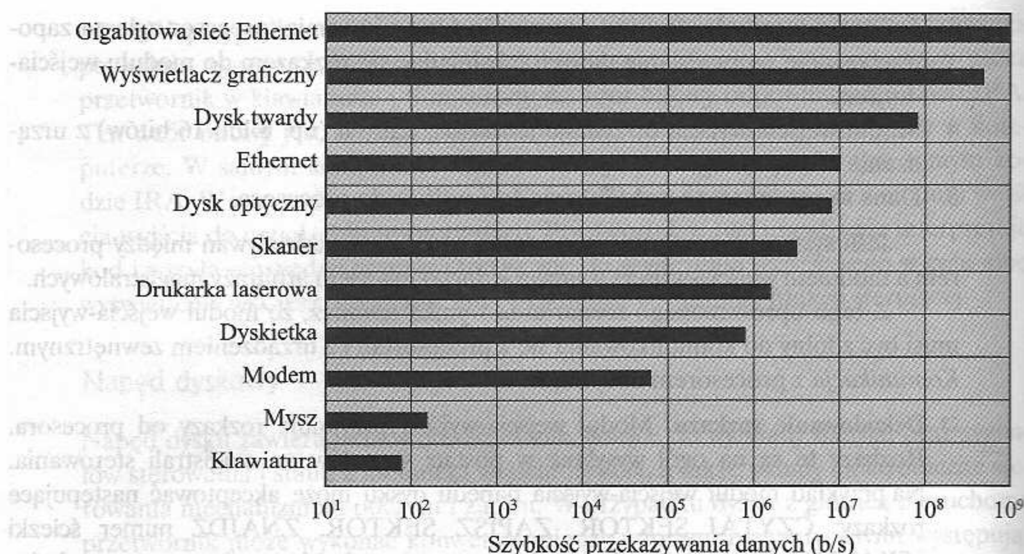
Jeśli system wykorzystuje magistralę, to każde z oddziaływań między procesorem a modułem wejścia-wyjścia wymaga jednego lub wielu arbitraży magistralowych.

Z tego uproszczonego scenariusza wynika również, że moduł wejścia-wyjścia musi być zdolny do komunikowania się z procesorem i z urządzeniem zewnętrznym. *Komunikacja* z procesorem obejmuje:

- **Dekodowanie rozkazu.** Moduł wejścia-wyjścia przyjmuje rozkazy od procesora. Rozkazy te są na ogół wysyłane w postaci sygnałów na magistrali sterowania. Na przykład moduł wejścia-wyjścia napędu dysku może akceptować następujące rozkazy: CZYTAJ SEKTOR, ZAPISZ SEKTOR, ZNAJDŹ numer ścieżki i SKANUJ rekord ID. Każdy z dwóch ostatnich rozkazów zawiera parametr, który jest przesyłany magistralą danych.
- **Przesyłanie danych.** Dane są wymieniane między modułem wejścia-wyjścia a procesorem poprzez magistralę danych.
- **Przesyłanie informacji o stanie.** Ponieważ urządzenia peryferyjne są tak powolne, ważna jest znajomość stanu modułu wejścia-wyjścia. Jeśli na przykład moduł wejścia-wyjścia jest proszony o przesłanie danych do procesora (odczyt), może on nie być gotowy do tego, ponieważ pracuje jeszcze nad poprzednim rozkazem wejścia-wyjścia. Fakt ten może być przedstawiony za pomocą sygnału stanu. Zwykłymi sygnałami stanu są sygnały zajętości (*busy*) i gotowości (*ready*). Mogą też występować sygnały informujące o różnych warunkach błędu.
- **Rozpoznawanie adresu.** Podobnie jak słowa w pamięci, również każde urządzenie wejścia-wyjścia ma swój adres. Moduł wejścia-wyjścia musi więc rozpoznawać unikatowy adres każdego spośród urządzeń peryferyjnych, którym steruje.

Moduł wejścia-wyjścia musi również **komunikować się** z urządzeniem. Komunikacja ta obejmuje rozkazy, informacje o stanie i dane (rys. 7.2).

Podstawowym zadaniem modułu wejścia-wyjścia jest **buforowanie danych**. Potrzeba tej funkcji jasno wynika z rys. 7.3. Podczas gdy szybkość transferu danych z (i do) pamięci głównej oraz procesora jest całkiem duża, w przypadku większości urządzeń peryferyjnych jest ona o rzędy wielkości mniejsza. Dane nadchodzące z pamięci głównej są przesyłane do modułu wejścia-wyjścia w postaci zwartego pakietu. Są one buforowane w module wejścia-wyjścia, a następnie wysyłane do urządzenia peryferyjnego z szybkością dostosowaną do tego urządzenia. Przy przesyłaniu w przeciwnym kierunku dane są buforowane, żeby nie angażować pamięci w operację powolnego transferu. Moduł wejścia-wyjścia musi więc móc pracować zarówno z szybkością urządzenia, jak i z szybkością pamięci. Podobnie, jeśli urządzenie wejścia-wyjścia funkcjonuje z szybkością większą w porównaniu z dostępem do pamięci, to moduł wejścia-wyjścia wykonuje niezbędną operację buforowania.



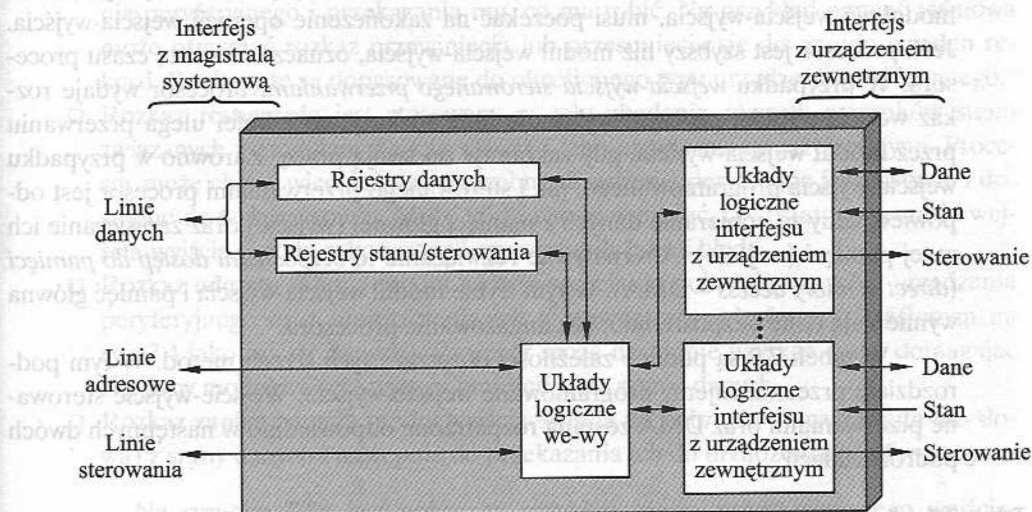
Rysunek 7.3. Typowe szybkości przekazywania danych urządzeń wejścia-wyjścia

Moduł wejścia-wyjścia jest często odpowiedzialny za **wykrywanie błędów** oraz za informowanie o nich procesora. Jedna z klas błędów obejmuje defekty mechaniczne i elektryczne występujące w urządzeniu (np. uszkodzenie papieru, niewłaściwa ścieżka dysku). Do innej klasy należą niezamierzone zmiany wzoru bitowego danych transmitowanych z urządzenia do modułu wejścia-wyjścia. Do wykrywania błędów transmisji często jest używana pewna postać kodu detekcyjnego. Typowym przykładem jest używanie bitu parzystości towarzyszącego każdemu znakowi danych. Na przykład kod znaku IRA zajmuje 7 bitów w bajcie. Bit ósmy jest ustalany na podstawie tego, czy suma wszystkich jedynek w bajcie jest parzysta, czy nieparzysta. Po odebraniu bajta moduł wejścia-wyjścia sprawdza parzystość w celu stwierdzenia, czy wystąpił błąd.

Struktura modułu wejścia-wyjścia

Moduły wejścia-wyjścia różnią się znacznie pod względem złożoności oraz liczby kontrolowanych urządzeń zewnętrznych. Możemy tu przedstawić jedynie bardzo ogólny opis (szczególne urządzenie, układ Intel 82C55A, jest opisane w podrozdz. 7.4). Na rysunku 7.4 jest przedstawiony ogólny schemat blokowy modułu wejścia-wyjścia. Moduł jest połączony z resztą komputera za pomocą zespołu linii sygnałowych (np. magistrali systemowej). Dane przenoszone do (i z) modułu są buforowane w jednym lub w wielu rejestrach danych. Może też występować jeden lub wiele rejestrów stanu, które dostarczają bieżącej informacji o stanie. Rejestr stanu może również funkcjonować jako rejestr sterowania, akceptujący szczegółową informację sterowania pochodzącą z procesora. Układy logiczne wewnątrz modułu współpracują z procesorem poprzez zespół linii sterowania. Są one wykorzystywane

przez procesor do wydawania rozkazów modułowi wejścia-wyjścia. Niektóre linie sterowania mogą być używane przez moduł wejścia-wyjścia (np. do przekazywania sygnałów arbitrażowych oraz określających stan). Moduł musi również rozpoznawać i generować adresy urządzeń, którymi steruje. Każdy moduł wejścia-wyjścia ma unikatowy adres lub, jeśli kontroluje więcej niż jedno urządzenie zewnętrzne, unikatowy zespół adresów. Moduł wejścia-wyjścia zawiera też układy logiczne dostosowane do interfejsów z każdym sterowanym urządzeniem.



Rysunek 7.4. Schemat blokowy modułu wejścia-wyjścia

Moduł wejścia-wyjścia funkcjonuje tak, żeby procesor mógł postrzegać szeroki zakres różnorodnych urządzeń zewnętrznych w prosty sposób. Istnieje wiele możliwości, którymi może dysponować moduł. Moduł wejścia-wyjścia może ukrywać szczegóły dotyczące taktowania, formatowania i działania elektromechanicznego urządzenia zewnętrznego, dzięki czemu procesor może się posługiwać prostymi rozkazami odczytu i zapisu, ewentualnie otwarcia i zamknięcia pliku. W najprostszej formie moduł wejścia-wyjścia może pozostawić wiele pracy dotyczącej sterowania urządzeniem (np. przewijanie taśmy) w gestii procesora.

Moduł wejścia-wyjścia, który przejmuje większość obciążenia szczegółowym przetwarzaniem, mający wysoki poziom priorytetu w stosunku do procesora, jest zwykle określany jako *kanał wejścia-wyjścia* lub *procesor wejścia-wyjścia*. Moduł wejścia-wyjścia, który jest całkiem prymitywny i wymaga szczegółowego sterowania, jest zwykle określany jako *sterownik wejścia-wyjścia* lub *sterownik urządzenia*. Sterowniki wejścia-wyjścia powszechnie występują w mikrokomputerach, podczas gdy kanały wejścia-wyjścia są wykorzystywane w dużych komputerach.

W dalszym ciągu będziemy stosowali termin **moduł wejścia-wyjścia**, gdy nie będzie to wywoływało nieporozumień, wykorzystując bardziej specyficzne określenia, jeśli będzie to konieczne.

7.3. Programowane wejście-wyjście

Istnieją trzy sposoby realizacji operacji wejścia-wyjścia. W przypadku *programowanego wejścia-wyjścia*, dane są wymieniane między procesorem a modułem wejścia-wyjścia. Procesor wykonuje program, który umożliwia mu bezpośrednie sterowanie operacją wejścia-wyjścia, włącznie z rozpoznawaniem stanu urządzenia, wysyłaniem rozkazu odczytu lub zapisu oraz transferem danych. Gdy procesor wydaje rozkaz modułowi wejścia-wyjścia, musi poczekać na zakończenie operacji wejścia-wyjścia. Jeśli procesor jest szybszy niż moduł wejścia-wyjścia, oznacza to stratę czasu procesora. W przypadku *wejścia-wyjścia sterowanego przerwaniami* procesor wydaje rozkaz wejścia-wyjścia, po czym wykonuje inne rozkazy, co z kolei ulega przerwaniu przez moduł wejścia-wyjścia, gdy zakończył on swoją pracę. Zarówno w przypadku wejścia-wyjścia programowanego, jak i sterowanego przerwaniami procesor jest odpowiedzialny za pobieranie danych z pamięci głównej (wyjście) oraz zapisywanie ich w tej pamięci (wejście). Alternatywne rozwiązanie to *bezpośredni dostęp do pamięci* (*direct memory access* – DMA). W tym trybie moduł wejścia-wyjścia i pamięć główna wymieniają dane bezpośrednio, bez angażowania procesora.

W tabeli 7.3 są podane zależności dotyczące tych trzech metod. W tym podrozdziale przeanalizujemy programowane wejście-wyjście. Wejście-wyjście sterowane przerwaniami oraz DMA zostaną rozpatrzone odpowiednio w następnych dwóch podrozdziałach.

Tabela 7.3. Sposoby działania wejścia-wyjścia

	Brak przerwań	Zastosowanie przerwań
Transfer wejście-wyjście – pamięć poprzez procesor	Programowane wejście-wyjście	Wejście-wyjście sterowane przerwaniami
Bezpośredni transfer wejście-wyjście – pamięć		Bezpośredni dostęp do pamięci (DMA)

Przegląd

Gdy procesor realizuje program i napotyka instrukcję odnoszącą się do wejścia-wyjścia, wykonuje tę instrukcję przez wydanie rozkazu odpowiedniemu modułowi wejścia-wyjścia. W przypadku programowanego wejścia-wyjścia moduł wejścia-wyjścia wykona wymagane działanie, a następnie ustawi odpowiednie bity w rejestrze stanu wejścia-wyjścia (rys. 7.4). Moduł wejścia-wyjścia nie podejmuje dalszych działań alarmujących procesor. W szczególności nie przerywa pracy procesora. Tak więc do procesora należy okresowe sprawdzanie stanu modułu wejścia-wyjścia i stwierdzenie, że operacja została zakończona.

Aby wyjaśnić działanie programowanego wejścia-wyjścia, rozpatrzymy je najpierw z punktu widzenia rozkazów wejścia-wyjścia wydawanych przez procesor modułowi wejścia-wyjścia, a następnie z punktu widzenia instrukcji wejścia-wyjścia wykonywanych przez procesor.

Rozkazy wejścia-wyjścia

W celu wykonania instrukcji odnoszącej się do wejścia-wyjścia procesor podaje adres określający moduł wejścia-wyjścia i urządzenie zewnętrzne oraz rozkaz wejścia-wyjścia. Istnieją cztery rodzaje rozkazów wejścia-wyjścia, które może otrzymać moduł wejścia-wyjścia adresowany przez procesor:

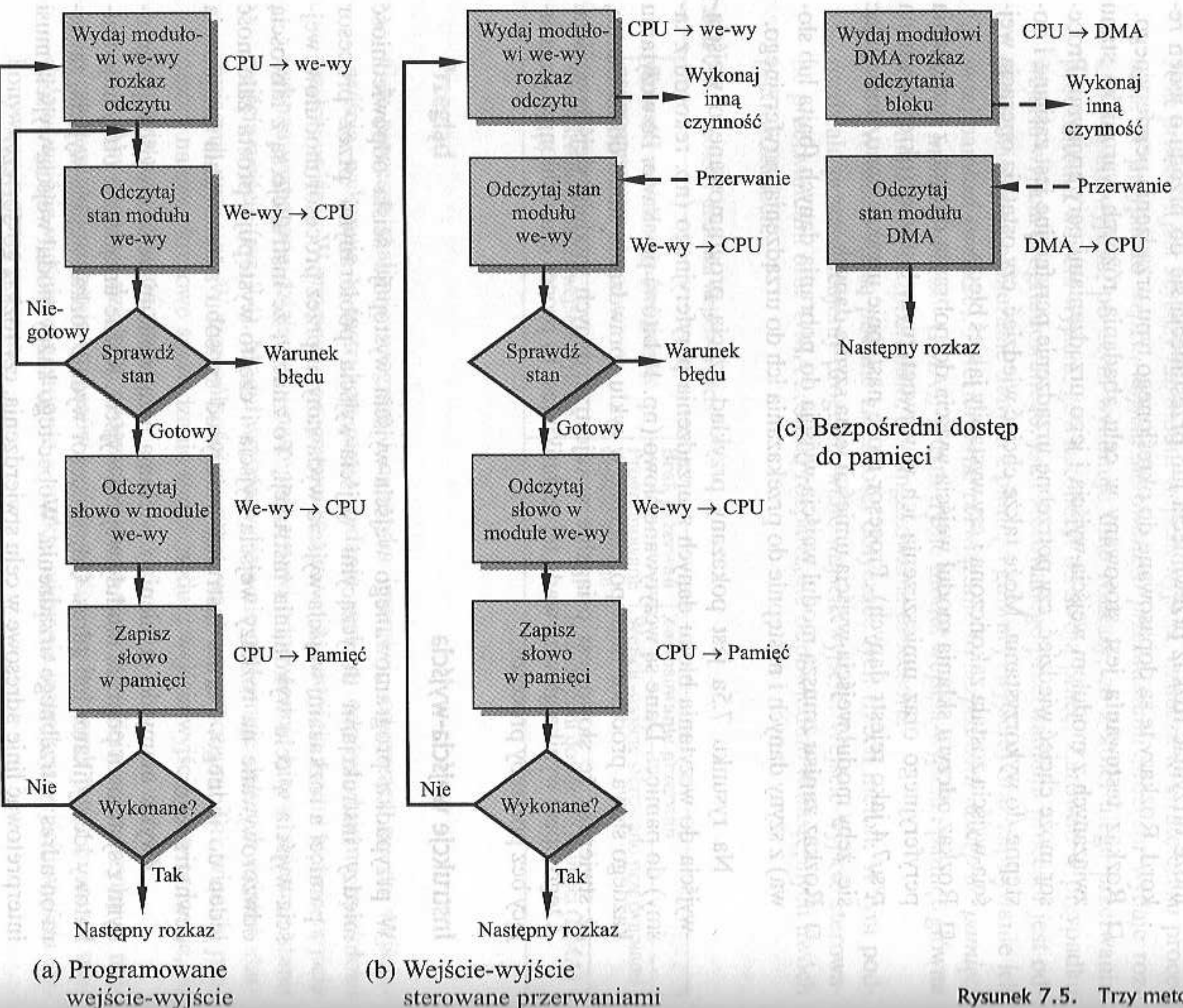
- Rozkaz **sterowania** jest stosowany w celu aktywowania (uruchomienia) urządzenia peryferyjnego i przekazania mu, co ma robić. Na przykład pamięć taśmowa może otrzymać rozkaz przewinięcia lub przesunięcia się do przodu o jeden rekord. Rozkazy te są dopasowane do określonego typu urządzenia peryferyjnego.
- Rozkaz **testowania** jest stosowany w celu zbadania różnych warunków stanu związanych z modułem wejścia-wyjścia i jego urządzeniami peryferyjnymi. Procesor może chcieć wiedzieć, czy potrzebne urządzenie peryferyjne jest zasilane i dostępne do wykorzystania. Może także chcieć wiedzieć, czy ostatnia operacja wejścia-wyjścia została zakończona i czy wystąpiły jakieś błędy.
- Rozkaz **odczytu** skłania moduł wejścia-wyjścia do pobrania danych z urządzenia peryferyjnego oraz umieszczenia ich w wewnętrznym buforze (określonym na rys. 7.4 jako rejestr danych). Procesor może następnie uzyskać dane, domagając się, żeby moduł wejścia-wyjścia umieścił je na szynie danych.
- Rozkaz **zapisu** zmusza moduł wejścia-wyjścia do pobrania danych (bajta lub słowa) z szyny danych i następnie do przekazania ich do urządzenia peryferyjnego.

Na rysunku 7.5a jest pokazany przykład użycia programowanego wejścia-wyjścia do wczytania bloku danych z urządzenia peryferyjnego (np. rekordu z taśmy) do pamięci. Dane są wczytywane słowo (np. 16 bitów) po słowie. Po wczytaniu każdego słowa procesor musi pozostawać w cyklu sprawdzania stanu do momentu, aż stwierdzi, że słowo jest osiągalne w rejestrze danych modułu wejścia-wyjścia. Z tego opisu wynika główna wada tej techniki: jest to proces czasochłonny, angażujący bez potrzeby procesor.

Instrukcje wejścia-wyjścia

W przypadku programowanego wejścia-wyjścia występuje ścisła odpowiedniość między instrukcjami dotyczącymi wejścia-wyjścia pobieranymi przez procesor z pamięci a rozkazami wejścia-wyjścia wydawanymi przez procesor modułowi wejścia-wyjścia w celu wykonania instrukcji. To znaczy, że instrukcje są z łatwością odwzorowywane na rozkazy wejścia-wyjścia i często występuje prosta zależność jeden do jednego. Forma instrukcji zależy od sposobu adresowania urządzenia zewnętrznego.

Zwykle mamy do czynienia z wieloma urządzeniami wejścia-wyjścia połączonymi z systemem poprzez moduły wejścia-wyjścia. Każde urządzenie otrzymuje unikatowy identyfikator lub adres. Gdy procesor wydaje rozkaz wejścia-wyjścia, zawiera on adres potrzebnego urządzenia. Wobec tego każdy moduł wejścia-wyjścia musi interpretować linie adresowe w celu stwierdzenia, czy rozkaz go dotyczy.

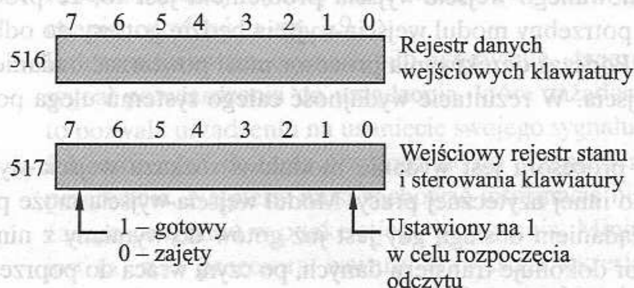


Rysunek 7.5. Trzy metody wprowadzania bloku danych

Jeśli procesor, pamięć główna oraz wejście-wyjście używają wspólnej magistrali, możliwe są dwa tryby adresowania: odwzorowany w pamięci i izolowany. W przypadku wejścia-wyjścia **odwzorowanego w pamięci** ta sama przestrzeń adresowa jest przeznaczona dla komórek pamięci i urządzeń wejścia-wyjścia. Procesor traktuje rejestry stanu i rejestry danych modułów wejścia-wyjścia jako komórki pamięci i wykorzystuje takie same instrukcje maszynowe w celu uzyskania dostępu zarówno do pamięci, jak i do urządzeń wejścia-wyjścia. Za pomocą na przykład 10 linii adresu możliwe jest obsługiwanie łącznie $2^{10} = 1024$ komórek pamięci i adresów wejścia-wyjścia, w dowolnej kombinacji.

W przypadku wejścia-wyjścia odwzorowanego w pamięci w magistrali potrzebna jest tylko jedna linia odczytu i jedna linia zapisu. Alternatywnie, magistrala może być wyposażona w linie odczytu i zapisu pamięci oraz linie rozkazów wejścia i wyjścia. Wówczas linia rozkazu określa, czy adres odnosi się do komórki pamięci, czy do urządzenia wejścia-wyjścia. Pełny zakres adresów może być osiągalny dla obu. Za pomocą 10 linii adresu system może teraz obsłużyć zarówno 1024 komórki pamięci, jak i 1024 adresy wejścia-wyjścia. Ponieważ przestrzeń adresowa wejścia-wyjścia jest odizolowana od przestrzeni adresowej pamięci, rozwiązanie takie jest określane jako **izolowane wejście-wyjście**.

Na rysunku 7.6 są przedstawione te dwie techniki programowanego wejścia-wyjścia. Na rysunku 7.6a widać, jak może postrzegać programista interfejs prostego urządzenia wejściowego (np. klawiatury) w przypadku wejścia-wyjścia odwzorowane-



(a)

ADRES	ROZKAZ	ARGUMENT	KOMENTARZ
200	Load AC	"1"	
	Store AC	517	Inicjuj odczyt klawiatury
202	Load AC	517	Uzyskaj bajt stanu
	Branch if Sign = 0	202	Wykonuj pętlę, aż do osiągnięcia gotowości
	Load AC	516	Ładuj bajt danych

(b)

ADRES	ROZKAZ	ARGUMENT	KOMENTARZ
200	StartI/O	5	Inicjuj odczyt klawiatury
201	Test I/O	5	Sprawdź zakończenie operacji
	Branch not ready	201	Wykonuj pętlę, aż do zakończenia operacji
	In	5	Ładuj bajt danych

Rysunek 7.6. Wejście-wyjście odwzorowane w pamięci (a) oraz izolowane (b)

go w pamięci. Załóżmy, że mamy adres 10-bitowy, z 512-bitową pamięcią (komórki 0÷511) i możliwymi 512 adresami wejścia-wyjścia (komórki 512÷1023). Dwa adresy są przypisane wejściu z klawiatury określonego terminalu. Adres 516 dotyczy rejestru danych, a adres 517 rejestru stanu, który działa również jako rejestr sterowania odbierający rozkazy procesora. Pokazany program powoduje wczytanie 1 bajta danych z klawiatury do rejestru akumulatora w procesorze. Zwróćmy uwagę, że procesor wykonuje pętlę do momentu, w którym bajt danych staje się osiągalny.

W przypadku izolowanego wejścia-wyjścia (rys. 7.6b) porty wejścia-wyjścia są dostępne tylko dla specjalnych rozkazów wejścia-wyjścia, które aktywują linie rozkazów wejścia-wyjścia magistrali.

W większości typów procesorów występuje stosunkowo duży zestaw różnych instrukcji dotyczących pamięci. Jeśli wykorzystuje się izolowane wejście-wyjście, to mamy do czynienia tylko z niewieloma instrukcjami wejścia-wyjścia. Wobec tego zaletą wejścia-wyjścia odwzorowanego w pamięci jest możliwość korzystania z obszernego zbioru instrukcji, co pozwala na wydajniejsze programowanie. Wadą jest zużywanie cennej przestrzeni adresowej pamięci. Powszechnie wykorzystuje się zarówno wejście-wyjście odwzorowane w pamięci, jak i izolowane.

7.4. Wejście-wyjście sterowane przerwaniami

W przypadku programowanego wejścia-wyjścia problemem jest to, że procesor musi długo czekać, aż potrzebny moduł wejścia-wyjścia będzie gotowy do odbioru lub transmisji danych. Podczas oczekiwania procesor musi powtarzać badanie stanu modułu wejścia-wyjścia. W rezultacie wydajność całego systemu ulega poważnej degradacji.

Alternatywą dla procesora jest wydanie modułowi rozkazu wejścia-wyjścia, a następnie przejście do innej użytecznej pracy. Moduł wejścia-wyjścia może przerwać pracę procesora żądaniem obsługi, gdy jest już gotów do wymiany z nim danych. Wówczas procesor dokonuje transferu danych, po czym wraca do poprzedniego przetwarzania.

Rozważmy, jak to funkcjonuje – najpierw z punktu widzenia modułu wejścia-wyjścia. W przypadku wejścia moduł wejścia-wyjścia otrzymuje od procesora rozkaz CZYTAJ. Następnie moduł wejścia-wyjścia przystępuje do odczytania danych ze stowarzyszonego urządzenia peryferyjnego. Gdy dane znajdują się już w rejestrze danych modułu, sygnalizuje on procesorowi przerwanie poprzez linię sterowania. Następnie moduł czeka, aż procesor zażąda danych. Gdy żądanie zostanie zgłoszone, moduł umieszcza dane na magistrali danych, po czym jest gotowy do innej operacji wejścia-wyjścia.

Z punktu widzenia procesora działanie w przypadku wejścia jest następujące. Procesor wydaje rozkaz CZYTAJ. Następnie przechodzi do innej czynności (może on jednocześnie realizować kilka różnych programów). Na końcu każdego cyklu rozkazu procesor sprawdza, czy nie nastąpiło przerwanie (rys. 3.9). Gdy następuje przerwanie ze strony modułu wejścia-wyjścia, procesor zachowuje kontekst bieżące-

go programu (np. zawartości licznika programu i rejestrów procesora) i przetwarza przerwanie. W tym przypadku procesor odczytuje słowo danych z modułu wejścia-wyjścia i kieruje je do pamięci. Następnie odnawia kontekst programu, nad którym pracował (lub jakiś inny program), i wznawia pracę.

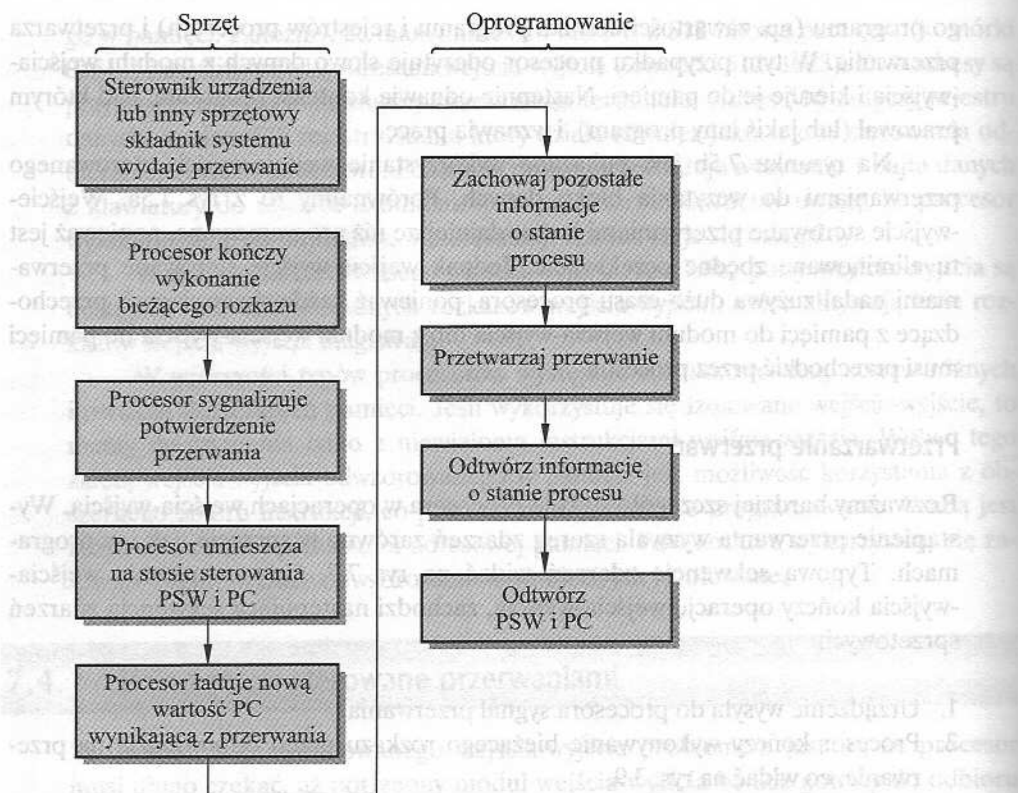
Na rysunku 7.5b jest pokazane wykorzystanie wejścia-wyjścia sterowanego przerwaniami do wczytania bloku danych. Porównajmy to z rys. 7.5a. Wejście-wyjście sterowane przerwaniami jest wydajniejsze niż programowane, ponieważ jest tu eliminowane zbędne oczekiwanie. Jednak wejście-wyjście sterowane przerwaniami nadal zużywa dużo czasu procesora, ponieważ każde słowo danych przechodzące z pamięci do modułu wejścia-wyjścia lub z modułu wejścia-wyjścia do pamięci musi przechodzić przez procesor.

Przetwarzanie przerwań

Rozważmy bardziej szczegółowo rolę procesora w operacjach wejścia-wyjścia. Wystąpienie przerwania wyzwala szereg zdarzeń zarówno w sprzęcie, jak i w programach. Typową sekwencję zdarzeń widać na rys. 7.7. Gdy urządzenie wejścia-wyjścia kończy operację wejścia-wyjścia, zachodzi następująca sekwencja zdarzeń sprzętowych:

1. Urządzenie wysyła do procesora sygnał przerwania.
2. Procesor kończy wykonywanie bieżącego rozkazu przed odpowiedzią na przerwanie, co widać na rys. 3.9.
3. Procesor sprawdza, czy nastąpiło przerwanie, stwierdza jego wystąpienie i wysyła sygnał potwierdzenia do urządzenia, które zażądało przerwania. Potwierdzenie to pozwala urządzeniu na usunięcie swojego sygnału przerwania.
4. Procesor musi się teraz przygotować do przeniesienia sterowania do procedury przerwania. Najpierw musi zachować informację niezbędną do wznowienia bieżącego programu w punkcie jego przerwania. Minimalną wymaganą informacją jest (a) stan procesora, zawarty w rejestrze określanym jako słowo stanu programu (*program status word* – PSW), oraz (b) lokalizacja następnego rozkazu przewidzianego do wykonania, która jest zawarta w liczniku programu. Informacja ta może być przekazana na stos sterowania systemu[3].
5. Procesor ładuje teraz do licznika rozkazów pierwszą pozycję programu obsługi przerwań, który odpowiada zgłoszonemu przerwaniu. Zależnie od architektury komputera i rozwiązania systemu operacyjnego może to być pojedynczy program, jeden dla każdego rodzaju przerwania, lub jeden dla każdego urządzenia i każdego typu przerwania. Jeśli występuje więcej niż jedna procedura obsługi przerwania, to procesor musi zdecydować, którą należy uruchomić. Informacja ta może być zawarta w oryginalnym sygnale przerwania lub też procesor może być zmuszony zwrócić się do urządzenia zgłaszającego przerwanie, aby w odpowiedzi uzyskać tę informację.

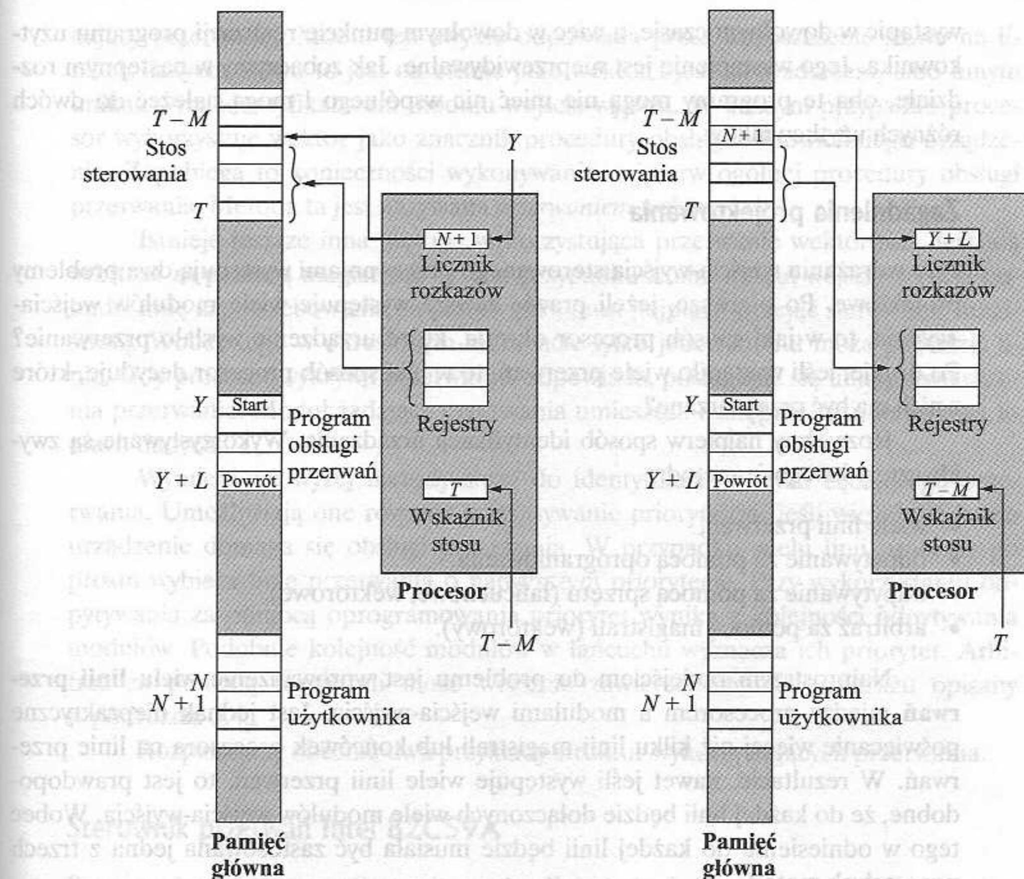
[3] Działanie stosu jest opisane w dodatku 10A.



Rysunek 7.7. Proste przetwarzanie przerwań

Gdy licznik rozkazów został załadowany, procesor przechodzi do następnego cyklu rozkazu, który rozpoczyna się od pobrania rozkazu. Ponieważ pobieranie rozkazów jest zdeterminowane przez zawartość licznika rozkazów, w wyniku tego sterowanie zostaje przeniesione do programu obsługi przerwań. Wynikiem realizowania tego programu są następujące operacje:

6. W tym punkcie zawartość licznika rozkazów oraz słowo stanu programu związane z przerwanym programem zostały zachowane na stosie systemowym. Są jednak również i inne informacje, które mogą być traktowane jako część „stanu" wykonywanego programu. W szczególności zachowania wymaga zawartość rejestrów procesora, ponieważ rejestry te mogą być wykorzystywane przez program obsługi przerwań. Wszystkie te wartości oraz inne informacje o stanie muszą być zachowane. Zwykle działanie programu obsługi przerwań rozpoczyna się od zapisania na stosie zawartości wszystkich rejestrów. Prosty przykład jest pokazany na rys. 7.8a. W tym przypadku program użytkownika został przerwany po instrukcji zawartej w pozycji N. Zawartość wszystkich rejestrów oraz adres następnego rozkazu ($N + 1$) są umieszczane na stosie. Wskaźnik wierzchołka stosu jest aktualizowany, aby wskazywał nowy wierzchołek, a licznik rozkazów jest aktualizowany, aby wskazywał początek procedury obsługi przerwania.



Rysunek 7.8. Zmiany w pamięci i w rejestrach wynikające z przerwania

7. Program obsługi przerwania może teraz przystąpić do przetwarzania przerwania. Obejmuje to zbadanie informacji stanu odnoszącej się do operacji wejścia-wyjścia lub innego zdarzenia, które spowodowało przerwanie. Może także obejmować wysłanie dodatkowych rozkazów lub potwierdzeń do urządzenia wejścia-wyjścia.
8. Gdy przetwarzanie przerwania jest zakończone, zachowane wartości rejestrów są pobierane ze stosu i wprowadzane do rejestrów (patrz np. rys. 7.8b).
9. Ostatnim działaniem jest odnowienie słowa stanu programu i licznika rozkazów na podstawie wartości ze stosu. W rezultacie następny rozkaz do realizacji będzie pochodził z uprzednio przerwanego programu.

Zauważmy, że ważne jest zachowanie wszystkich informacji stanu dotyczących przerwanego programu w celu jego późniejszego przywrócenia. Dzieje się tak dlatego, że przerwanie nie jest procedurą wywołaną z tego programu. Przerwanie może

wystąpić w dowolnym czasie, a więc w dowolnym punkcie realizacji programu użytkownika. Jego wystąpienie jest nieprzewidywalne. Jak zobaczymy w następnym rozdziale, oba te programy mogą nie mieć nic wspólnego i mogą należeć do dwóch różnych użytkowników.

Zagadnienia projektowania

Przy wdrażaniu wejścia-wyjścia sterowanego przerwaniami występują dwa problemy projektowe. Po pierwsze, jeżeli prawie zawsze występuje wiele modułów wejścia-wyjścia, to w jaki sposób procesor określa, które urządzenie wysłało przerwanie? Po drugie, jeśli wystąpiło wiele przerwań, to w jaki sposób procesor decyduje, które z nich ma być przetwarzane?

Rozważmy najpierw sposób identyfikacji urządzenia. Wykorzystywane są zwykle cztery kategorie metod:

- wiele linii przerwań,
- odpytywanie za pomocą oprogramowania,
- odpytywanie za pomocą sprzętu (łańcuchowe, wektorowe),
- arbitraż za pomocą magistrali (wektorowy).

Najprostszym podejściem do problemu jest wprowadzenie **wielu linii przerwań** między procesorem a modułami wejścia-wyjścia. Jest jednak niepraktyczne poświęcanie więcej niż kilku linii magistrali lub końcówek procesora na linie przerwań. W rezultacie, nawet jeśli występuje wiele linii przerwań, to jest prawdopodobne, że do każdej linii będzie dołączonych wiele modułów wejścia-wyjścia. Wobec tego w odniesieniu do każdej linii będzie musiała być zastosowana jedna z trzech pozostałych metod.

Jedną z możliwości jest **odpytywanie za pomocą oprogramowania**. Gdy procesor wykrywa przerwanie, przeskakuje do procedury obsługi przerwania, której zadaniem jest odpytanie każdego modułu wejścia-wyjścia w celu stwierdzenia, który moduł spowodował przerwanie. Odpytywanie może być przeprowadzone za pomocą oddzielnej linii sterowania (np. TEST I/O). W tym przypadku procesor wzbudza TEST I/O i umieszcza adres określonego modułu wejścia-wyjścia na liniach adresu. Moduł wejścia-wyjścia odpowiada pozytywnie, jeśli wysłał przerwanie. Alternatywnie, każdy moduł wejścia-wyjścia może zawierać adresowalny rejestr stanu. Procesor odczytuje wtedy rejestr stanu każdego modułu wejścia-wyjścia w celu zidentyfikowania modułu przerywającego. Gdy właściwy moduł jest zidentyfikowany, procesor przeskakuje do procedury obsługi urządzenia właściwej dla tego urządzenia.

Wadą odpytywania za pomocą oprogramowania jest jego czasochłonność. Bardziej wydajną metodą jest wykorzystanie rozwiązania **łańcuchowego**, które w istocie jest odpytywaniem za pomocą sprzętu. Przykład konfiguracji łańcuchowej widać na rys. 3.26. Wszystkie moduły wejścia-wyjścia są połączone ze wspólną linią żądania przerwania. Linia potwierdzenia przerwania przechodzi łańcuchowo przez moduły. Gdy procesor wykrywa przerwanie, wysyła potwierdzenie przerwania. Sygnał ten przechodzi przez kolejne moduły wejścia-wyjścia, aż natrafi na moduł żą-

dający przerwania. Moduł ten zwykle odpowiada przez umieszczenie słowa na liniach danych. Słowo to jest określane jako wektor i jest albo adresem, albo innym unikatowym identyfikatorem modułu wejścia-wyjścia. W każdym przypadku procesor wykorzystuje wektor jako znacznik procedury obsługi odpowiedniego urządzenia. Zapobiega to konieczności wykonywania najpierw ogólnej procedury obsługi przerwania. Metoda ta jest nazywana *przerwaniem wektorowym*.

Istnieje jeszcze inna metoda wykorzystująca przerwanie wektorowe. Jest nią **arbitraż za pomocą magistrali**. W tym przypadku zanim moduł wejścia-wyjścia pobudzi linię zapotrzebowania na przerwanie, musi najpierw przejąć sterowanie magistralą. Wobec tego w określonym momencie tylko jeden moduł może pobudzić linię. Gdy procesor wykrywa przerwanie, odpowiada, posługując się linią potwierdzenia przerwania. Moduł żądający przerwania umieszcza następnie swój wektor na liniach danych.

Wymienione wyżej metody służą do identyfikacji modułu żądającego przerwania. Umożliwiają one również przypisywanie priorytetów, jeśli więcej niż jedno urządzenie domaga się obsługi przerwania. W przypadku wielu linii procesor po prostu wybiera linię przerwania o najwyższym priorytecie. Przy wykorzystaniu odpytywania za pomocą oprogramowania priorytet wynika z kolejności odpytywania modułów. Podobnie kolejność modułów w łańcuchu wyznacza ich priorytet. Arbitraż za pomocą magistrali może wreszcie zawierać schemat arbitrażu opisany w podrozdz. 3.4.

Rozpatrzymy obecnie dwa przykłady struktur wykorzystujących przerwania.

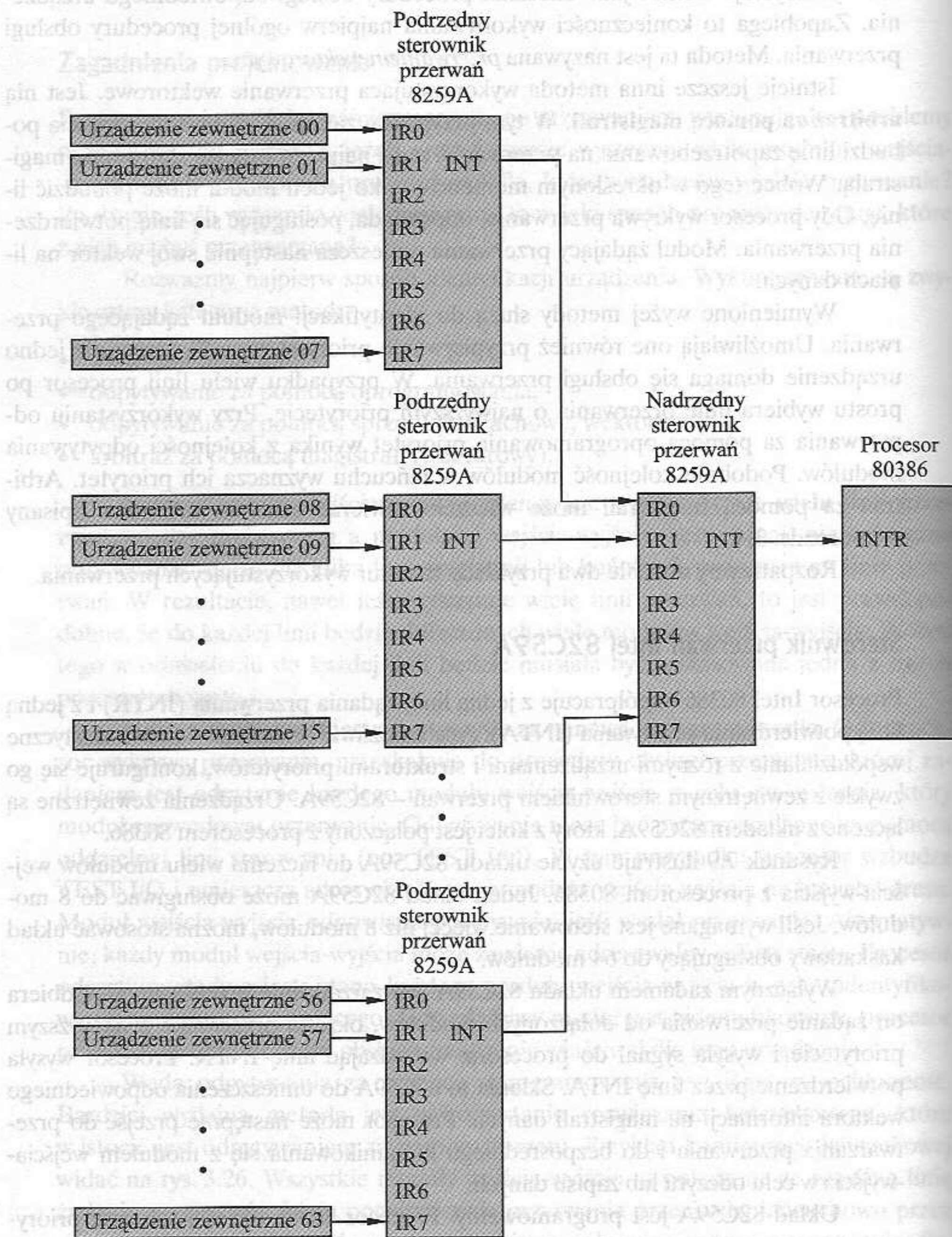
Sterownik przerwań Intel 82C59A

Procesor Intel 80386 współpracuje z jedną linią żądania przerwania (INTR) i z jedną linią potwierdzania przerwania (INTA). Aby umożliwić procesorowi 80386 elastyczne współdziałanie z różnymi urządzeniami i strukturami priorytetów, konfiguruje się go zwykle z zewnętrznym sterownikiem przerwań – 82C59A. Urządzenia zewnętrzne są łączone z układem 82C59A, który z kolei jest połączony z procesorem 80386.

Rysunek 7.9 ilustruje użycie układu 82C59A do łączenia wielu modułów wejścia-wyjścia z procesorem 80386. Jeden układ 82C59A może obsługiwać do 8 modułów. Jeśli wymagane jest sterowanie więcej niż 8 modułów, można stosować układ kaskadowy obsługujący do 64 modułów.

Wyłącznym zadaniem układu 82C59A jest zarządzanie przerwaniami. Odbiera on żądanie przerwania od dołączonych modułów, określa przerwanie o najwyższym priorytecie i wysyła sygnał do procesora, wzbudzając linię INTR. Procesor wysyła potwierdzenie przez linię INTA. Skłania to 82C59A do umieszczenia odpowiedniego wektora informacji na magistrali danych. Procesor może następnie przejść do przetwarzania przerwania i do bezpośredniego komunikowania się z modułem wejścia-wyjścia w celu odczytu lub zapisu danych.

Układ 82C59A jest programowalny. Procesor 80386 określa schemat priorytetów, który ma być wykorzystywany przez ustalenie słowa sterowania w sterowniku 82C59A. Możliwe są następujące tryby przerwania.

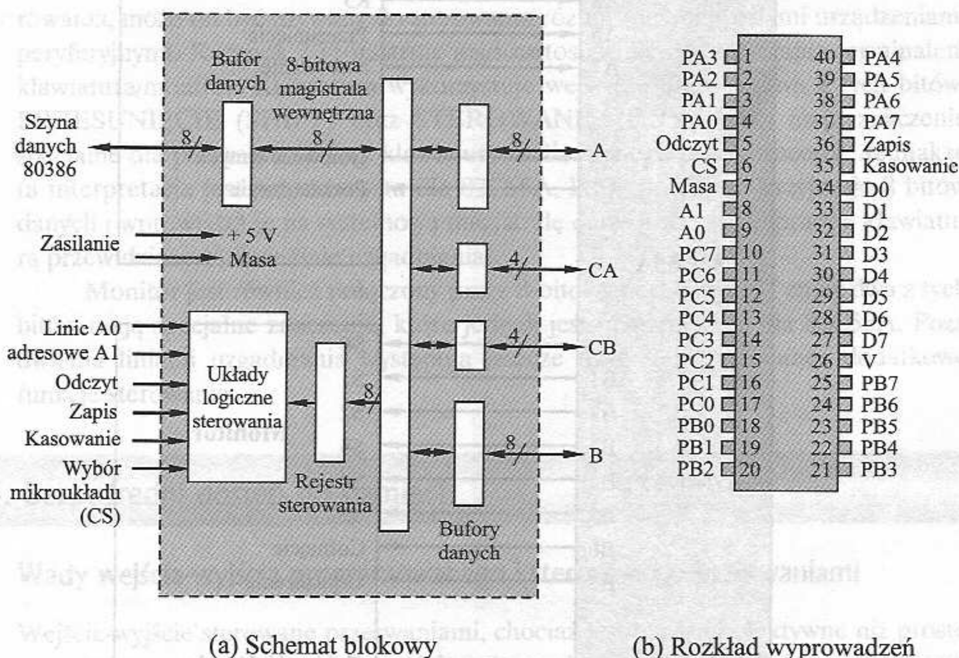


Rysunek 7.9. Zastosowanie sterownika przerwań 82C59A

- **W pełni zagnieżdżony.** Zapotrzebowania na przerwanie są porządkowane pod względem priorytetu od 0 (IR0) do 7 (IR7).
- **Rotacyjny.** W wielu zastosowaniach pewna liczba urządzeń przerywających ma jednakowy priorytet. W tym trybie urządzenie, którego obsługę zakończono, otrzymuje najniższy priorytet w grupie.
- **Ze specjalną maską.** Pozwala on procesorowi na selektywne wzbranianie przerwań od niektórych urządzeń.

Programowalny interfejs urządzeń peryferyjnych Intel 82C55A

Jako przykład modułu wejścia-wyjścia wykorzystywanego przez programowane wejście-wyjście oraz wejście-wyjście sterowane przerwaniami rozpatrzymy programowalny interfejs urządzeń peryferyjnych Intel 82C55A. Układ ten jest jednostrukturowym modułem wejścia-wyjścia o ogólnym przeznaczeniu, zaprojektowanym do wykorzystywania z procesorem Intel 80386. Na rysunku 7.10 widać jego ogólny schemat blokowy oraz przyporządkowanie wyprowadzeń obudowy o 40 końcówkach, w której jest zamykany.



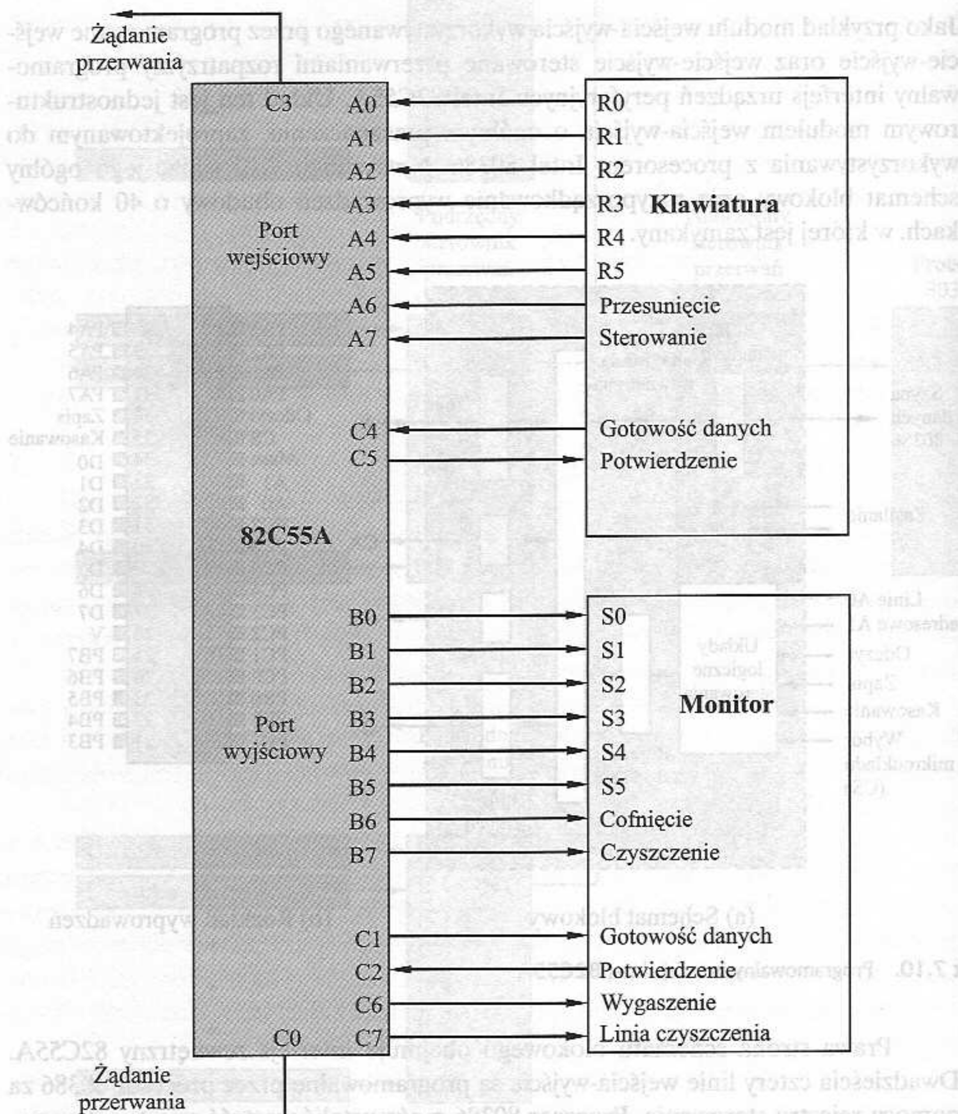
(a) Schemat blokowy (b) Rozkład wyprowadzeń

Rysunek 7.10. Programowalny interfejs Intel 82C55A

Prawa strona schematu blokowego obejmuje interfejs zewnętrzny 82C55A. Dwadzieścia cztery linie wejścia-wyjścia są programowalne przez procesor 80386 za pomocą rejestru sterowania. Procesor 80386 może ustalić wartość rejestru sterowania w celu sprecyzowania różnych trybów działania i konfiguracji. Linie wejścia-

-wyjścia są podzielone na trzy grupy 8-bitowe (A, B i C). Każda grupa może funkcjonować jako 8-bitowy port wejścia-wyjścia. Ponadto grupa C jest podzielona na grupy 4-bitowe (CA i CB), które mogą być używane w połączeniu z portami A i B. Przy takiej konfiguracji mogą one przenosić sygnały sterowania i stanu.

Lewą stronę schematu blokowego stanowi wewnętrzny interfejs do magistrali procesora 80386. Obejmuje on 8-bitową dwukierunkową szynę danych (D0÷D7) używaną do przesyłania danych do (i z) portów wejścia-wyjścia oraz do transferu informacji sterowania do rejestru sterowania. Dwie linie adresowe określają jeden z trzech portów wejścia-wyjścia lub rejestru sterowania. Transfer następuje, gdy zezwala na to



Rysunek 7.11. Połączenie klawiatury i monitora z układem 82C55A

stan linii wyboru mikroukładu (CHIP SELECT) oraz stan jednej z linii odczytu (READ) lub zapisu (WRITE). Linia RESET jest wykorzystywana do ustawiania modułu w stanie początkowym.

Rejestr sterowania jest ładowany przez procesor w celu sterowania trybem działania oraz definiowania sygnałów, jeśli jest taka potrzeba. W trybie działania 0 trzy grupy po 8 linii zewnętrznych funkcjonują jako trzy 8-bitowe porty wejścia-wyjścia. Każdy z portów może być wyznaczony jako wejściowy lub wyjściowy. W innym wypadku grupy A i B funkcjonują jako porty wejścia-wyjścia, natomiast linie z grupy C służą jako linie sterujące dla A i B. Sygnały sterowania służą dwóm zasadniczym celom: uzgadnianiu i żądaniu przerwań. Uzgadnianie wykorzystuje prosty mechanizm taktowania. Jedna linia sterowania jest używana przez nadawcę jako linia gotowości danych (DATA READY), wskazująca, kiedy dane są obecne na liniach danych wejścia-wyjścia. Inna linia jest używana przez odbiorcę do potwierdzania (ACKNOWLEDGE), wskazując, że dane zostały odczytane i linie danych mogą być wyczyszczone. Jeszcze inna linia może być oznaczona jako linia zapotrzebowania przerwania (INTERRUPT REQUEST) i połączona z magistralą systemową.

Ponieważ układ 82C55A jest programowalny za pośrednictwem rejestru sterowania, może on być używany do sterowania różnorodnymi prostymi urządzeniami peryferyjnymi. Rysunek 7.11 ilustruje jego zastosowanie do sterowania terminalem klawiatura/monitor. Klawiatura wykorzystuje wejście 8-bitowe. Dwa z tych bitów, PRZESUNIĘCIE (SHIFT) oraz STEROWANIE (CONTROL) mają znaczenie specjalne dla programu obsługi klawiatury realizowanego przez procesor. Jednakże ta interpretacja jest przezroczysta dla 82C55A, który po prostu przyjmuje 8 bitów danych i wprowadza je na systemową magistralę danych. Do współpracy z klawiaturą przewidziane są dwie linie uzgadniania.

Monitor jest również połączony przez 8-bitowy port danych. I znów dwa z tych bitów mają specjalne znaczenie, które jednak jest przezroczyste dla 82C55A. Poza dwiema liniami uzgadniania występują jeszcze dwie linie realizujące dodatkowe funkcje sterowania.

7.5. Bezpośredni dostęp do pamięci

Wady wejścia-wyjścia programowanego i sterowanego przerwaniami

Wejście-wyjście sterowane przerwaniami, chociaż jest bardziej efektywne niż proste programowane wejście-wyjście, nadal wymaga aktywnej interwencji procesora przy przesyłaniu danych między pamięcią a modułem wejścia-wyjścia, a każdy transfer danych musi wędrować drogą wiodącą przez procesor. A więc obie te formy wejścia-wyjścia mają dwie nieodłączne wady:

1. Szybkość transferu wejścia-wyjścia jest ograniczana szybkością, z jaką procesor może testować i obsługiwać urządzenie.

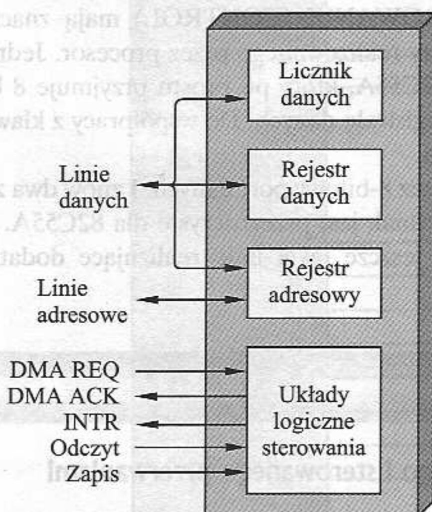
2. Procesor jest zajęty zarządzaniem przesyłaniem z wejścia i na wyjście; w przypadku każdego takiego transferu musi być wykonana pewna liczba rozkazów (patrz np. rys. 7.5).

Między tymi niedogodnościami występuje pewna wymienność. Rozważmy przesyłanie bloku danych. W przypadku prostego programowanego wejścia-wyjścia procesor jest przypisany określonemu zadaniu i może przenosić dane ze stosunkowo dużą szybkością, kosztem rezygnacji z innych zadań. Wejście-wyjście sterowane przerwaniami uwalnia do pewnego stopnia procesor, kosztem szybkości przesyłania. Niezależnie od tego obie metody mają ujemny wpływ zarówno na pracę procesora, jak i na szybkość przesyłania z wejścia i na wyjście.

Gdy muszą być przenoszone duże ilości danych, wymagana jest bardziej efektywna metoda: bezpośredni dostęp do pamięci (DMA).

Działanie DMA

Bezpośredni dostęp do pamięci wymaga dodatkowego modułu na magistrali systemowej. Moduł DMA (rys. 7.12) może „udawać" procesor i w rzeczywistości przejmować od procesora sterowanie systemem. Musi to robić, aby przenosić dane do pamięci (i z pamięci) poprzez magistralę systemową. W tym celu moduł DMA musi



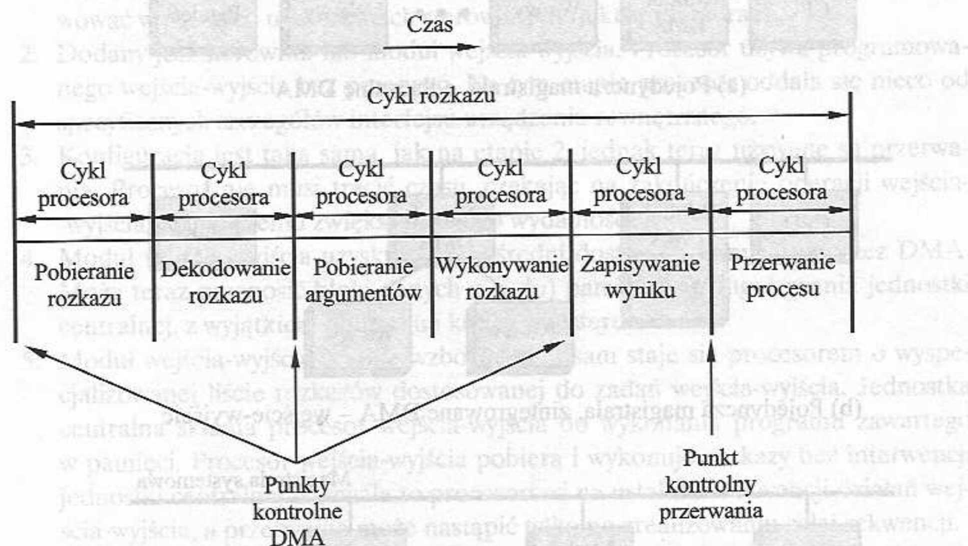
Rysunek 7.12. Schemat blokowy typowego modułu DMA

posługiwać się magistralą tylko wówczas, gdy procesor jej nie potrzebuje, lub zmusić procesor do czasowego zawieszenia działania. Ta ostatnia technika jest bardziej rozpowszechniona i jest określana jako *kradzież cyklu*, ponieważ w efekcie moduł DMA „wykrada" cykl magistrali.

Gdy procesor życzy sobie odczytania lub zapisania bloku danych, wydaje rozkaz modułowi DMA, wysyłając mu następujące informacje:

- czy wymagany jest odczyt, czy zapis, posługując się linią sterowania odczytem lub zapisem między procesorem a modułem DMA;
- adres niezbędnego urządzenia wejścia-wyjścia, komunikowany poprzez linie danych;
- adres początkowej komórki pamięci przewidzianej do odczytania lub do zapisania, komunikowany poprzez linie danych i zapisywany przez moduł DMA w rejestrze adresu tego modułu;
- liczbę słów, które mają być odczytane lub zapisane, komunikowaną poprzez linie danych i zapisywaną w rejestrze licznika danych.

Procesor następnie kontynuuje inne prace. Zleca tę operację wejścia-wyjścia modułowi DMA i moduł ten zatroszczy się o nią. Moduł DMA przenosi cały blok danych, słowo po słowie, bezpośrednio z (lub do) pamięci, bez przechodzenia przez procesor. Gdy transfer jest zakończony, moduł DMA wysyła sygnał przerwania do procesora. Dzięki temu procesor jest angażowany tylko na początku i na końcu transferu (rys. 7.5c).



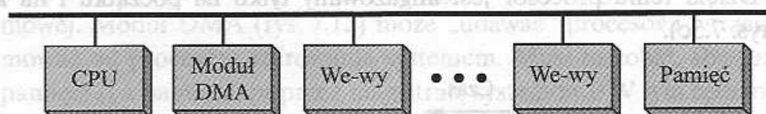
Rysunek 7.13. Punkty kontrolne DMA i przerwania podczas cyklu rozkazu

Na rysunku 7.13 są pokazane miejsca, w których może być zawieszane wykonanie cyklu rozkazu procesora. W każdym wypadku procesor jest zawieszany tuż przed zgłoszeniem zapotrzebowania na magistralę. Moduł DMA przesyła następnie jedno słowo i zwraca sterowanie do procesora. Zauważmy, że nie jest to przerwanie; procesor nie zachowuje kontekstu i nie wykonuje niczego innego. Procesor zatrzymuje się na jeden cykl magistrali. W ogólnym rozrachunku procesor pracuje nieco wolniej. Mimo to, w przypadku transferu z wejścia lub na wyjście obejmującego

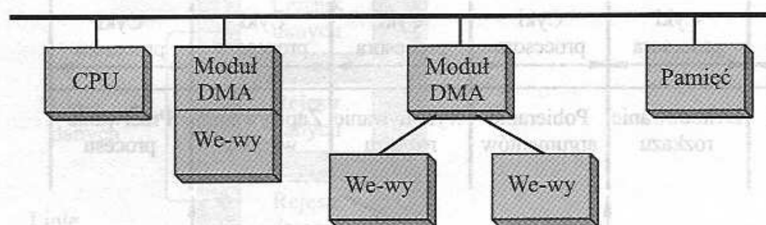
wiele słów, metoda DMA jest daleko bardziej efektywna niż wejście-wyjście sterowane przerwaniami lub programowane.

Mechanizm DMA może być konfigurowany na różne sposoby. Niektóre możliwości są pokazane na rys. 7.14. W pierwszym przykładzie wszystkie moduły używają tej samej magistrali systemowej. Moduł DMA, działając jako namiastka procesora, używa programowanego wejścia-wyjścia do wymiany danych między pamięcią a modułem wejścia-wyjścia za pośrednictwem modułu DMA. Konfiguracja ta, chociaż może być niedroga, jest wyraźnie nieefektywna. Podobnie jak w przypadku programowanego wejścia-wyjścia sterowanego przez procesor, każdy transfer słowa pochłania dwa cykle magistrali.

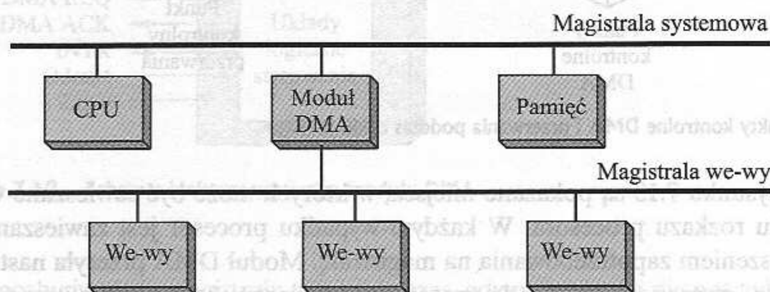
Liczba wymaganych cykli magistrali może być istotnie ograniczona przez zintegrowanie funkcji DMA i wejścia-wyjścia. Jak widać na rys. 7.14b, będzie wówczas istniała ścieżka między modułem DMA a jednym lub wieloma modułami wejścia-wyjścia nie angażująca magistrali systemowej. W istocie układ logiczny DMA może być częścią modułu wejścia-wyjścia lub może być oddzielnym modułem, który steruje jednym lub wieloma modułami wejścia-wyjścia. Koncepcja ta może być rozwinięta przez połą-



(a) Pojedyncza magistrala, odłączalne DMA



(b) Pojedyncza magistrala, zintegrowane DMA – wejście-wyjście



(c) Magistrala wejścia-wyjścia

Rysunek 7.14. Możliwe konfiguracje modułu DMA

czenie modułów wejścia-wyjścia z modułem DMA za pomocą magistrali wejścia-wyjścia (rys. 7.14c). Redukuje to liczbę interfejsów wejścia-wyjścia w module DMA do jednego i stanowi łatwo rozszerzalną konfigurację. We wszystkich tych przypadkach (rys. 7.14b i c) magistrala systemowa używana wspólnie przez moduł DMA, procesor i pamięć jest wykorzystywana przez DMA tylko do wymiany danych z pamięcią. Wymiana danych między DMA a modułami wejścia-wyjścia zachodzi poza magistralą systemową.

7.6. Kanały i procesory wejścia-wyjścia

Ewolucja funkcjonowania wejścia-wyjścia

W miarę ewolucji systemów komputerowych mieliśmy do czynienia z rosnącą złożonością i wyrafinowaniem poszczególnych zespołów. Nigdzie nie jest to bardziej widoczne niż w przypadku funkcjonowania wejścia-wyjścia. Omówiliśmy już część tej ewolucji. Jej etapy można podsumować następująco:

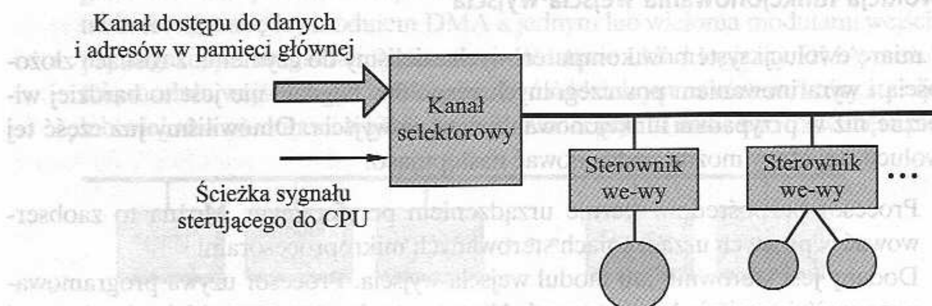
1. Procesor bezpośrednio steruje urządzeniem peryferyjnym. Można to zaobserwować w prostych urządzeniach sterowanych mikroprocesorami.
2. Dodany jest sterownik lub moduł wejścia-wyjścia. Procesor używa programowanego wejścia-wyjścia bez przerwań. Na tym etapie procesor oddala się nieco od specyficznych szczegółów interfejsu urządzenia zewnętrznego.
3. Konfiguracja jest taka sama, jak na etapie 2, jednak teraz używane są przerwania. Procesor nie musi tracić czasu, czekając na zakończenie operacji wejścia-wyjścia, dzięki czemu zwiększa się jego wydajność.
4. Moduł wejścia-wyjścia uzyskuje bezpośredni dostęp do pamięci poprzez DMA. Może teraz przenosić bloki danych z (i do) pamięci bez angażowania jednostki centralnej, z wyjątkiem początku i końca transferu.
5. Moduł wejścia-wyjścia zostaje wzbogacony i sam staje się procesorem o wyspecjalizowanej liście rozkazów dostosowanej do zadań wejścia-wyjścia. Jednostka centralna skłania procesor wejścia-wyjścia do wykonania programu zawartego w pamięci. Procesor wejścia-wyjścia pobiera i wykonuje rozkazy bez interwencji jednostki centralnej. Pozwala to procesorowi na ustalanie sekwencji działań wejścia-wyjścia, a przerwanie może nastąpić tylko po zrealizowaniu całej sekwencji.
6. Moduł wejścia-wyjścia uzyskuje własną pamięć i w rzeczywistości sam staje się komputerem. W przypadku tej architektury dużym zestawem urządzeń wejścia-wyjścia można sterować przy minimalnym zaangażowaniu jednostki centralnej. Powszechnie wykorzystuje się taką architekturę do sterowania komunikacją z terminalami konwersacyjnymi. Procesor wejścia-wyjścia przejmuje większość zadań związanych ze sterowaniem terminalami.

W miarę posuwania się po tej drodze ewolucji coraz więcej funkcji wejścia-wyjścia realizuje się bez angażowania procesora, który jest stopniowo uwalniany od zadań związanych z wejściem-wyjściem, co zwiększa jego wydajność. Na dwóch ostat-

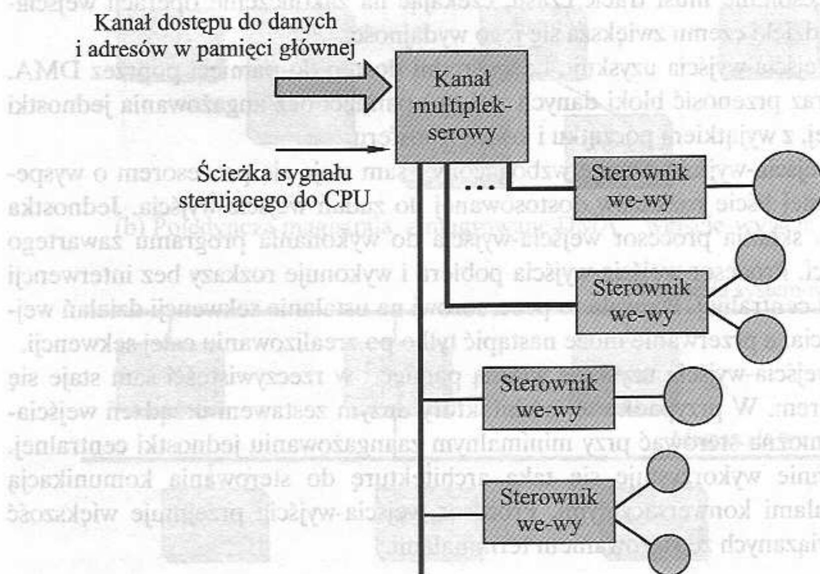
nich etapach (5 i 6) następuje poważna zmiana, polegająca na wprowadzeniu koncepcji modułu wejścia-wyjścia zdolnego do wykonywania programu. Na etapie 5 moduł wejścia-wyjścia jest często określany jako *kanał wejścia-wyjścia*. Na etapie 6 jest stosowany termin *procesor wejścia-wyjścia*. Czasem oba terminy bywają stosowane w obu sytuacjach. W dalszym ciągu będziemy posługiwali się terminem kanał wejścia-wyjścia.

Właściwości kanałów wejścia-wyjścia

Kanał wejścia-wyjścia stanowi rozszerzenie koncepcji DMA. Może on wykonywać rozkazy wejścia-wyjścia, a co za tym idzie – w pełni sterować operacjami wejścia-wyjścia. W systemie komputerowym wyposażonym w takie urządzenie jednostka cen-



(a) Selektor



(b) Multiplekser

Rysunek 7.15. Architektura kanału wejścia-wyjścia

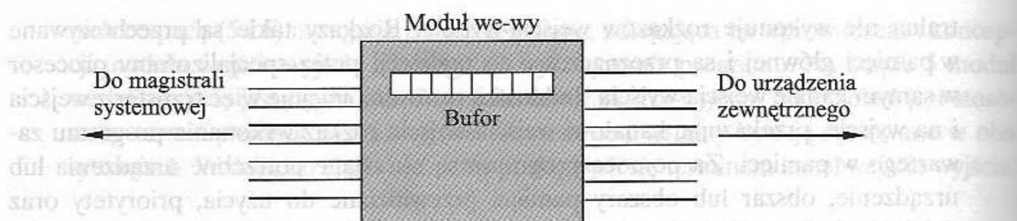
tralna nie wykonuje rozkazów wejścia-wyjścia. Rozkazy takie są przechowywane w pamięci głównej i są przeznaczone do realizacji przez specjalizowany procesor w samym kanale wejścia-wyjścia. Jednostka centralna inicjuje więc transfer z wejścia i na wyjście, przekazując kanałowi wejścia-wyjścia rozkaz wykonania programu zawartego w pamięci. Za pomocą programu są określane potrzebne urządzenia lub urządzenie, obszar lub obszary pamięci przewidziane do użycia, priorytety oraz działania, które mają być podejmowane w określonych warunkach wystąpienia błędu. Kanał wejścia-wyjścia wykonuje te rozkazy i steruje przesyłaniem danych.

Powszechnie spotykane są dwa typy kanałów wejścia-wyjścia, zilustrowane na rys. 7.15. *Kanał wybiórczy* (*selector channel*) steruje wieloma szybkimi urządzeniami i w określonym czasie zajmuje się transferem danych z jednego spośród tych urządzeń. Kanał wejścia-wyjścia wybiera więc urządzenie i realizuje transfer danych. Każde urządzenie lub niewielki zespół urządzeń jest sterowany za pomocą *sterownika* lub modułu wejścia-wyjścia, podobnego do modułów wejścia-wyjścia omawianych powyżej. Tak więc kanał wejścia-wyjścia zastępuje procesor w czynności kontrolowania tych sterowników wejścia-wyjścia. *Kanał multiplekserowy* (*multiplexor channel*) może jednocześnie współpracować z wieloma urządzeniami wejścia-wyjścia. W przypadku urządzeń powolnych *multiplekser bajtowy* odbiera lub przekazuje znaki do wielu urządzeń tak szybko, jak tylko jest to możliwe. Na przykład wynikowy strumień znaków pochodzących od trzech urządzeń o różnych szybkościach, wysyłających indywidualne strumienie A_1 A_2 A_3 A_4..., B_1 B_2 B_3 B_4... i C_1 C_2 C_3 C_4..., może wyglądać następująco: A_1 B_1 C_1 A_2 C_2 A_3 B_2 C_3 A_4 itd. W przypadku urządzeń szybkich *multiplekser blokowy* przeplata bloki danych z kilku urządzeń.

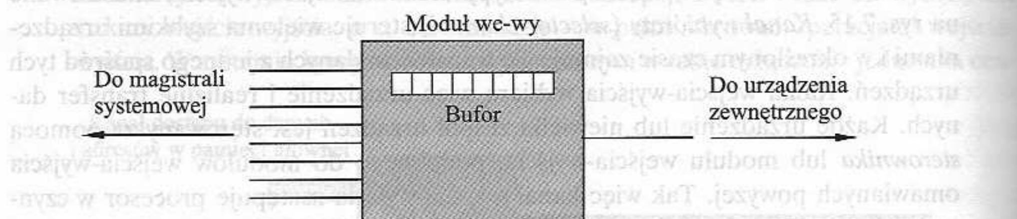
7.7. Interfejs zewnętrzny: FireWire i InfiniBand

Rodzaje interfejsów

Interfejs między urządzeniem peryferyjnym a modułem wejścia-wyjścia musi być dostosowany do natury i działania urządzenia peryferyjnego. Jedną z głównych cech tego interfejsu jest to, czy jest on szeregowy, czy równoległy (rys. 7.16). W **interfejsie równoległym** występuje wiele linii łączących moduł wejścia-wyjścia z urządzeniem peryferyjnym i wiele bitów jest przesyłanych jednocześnie, podobnie jak wszystkie bity słowa są przenoszone równocześnie przez szynę danych. W **interfejsie szeregowym** do transmisji danych służy tylko jedna linia i w określonym momencie jest transmitowany tylko jeden bit. Interfejs równoległy jest powszechnie wykorzystywany do współpracy z szybkimi urządzeniami peryferyjnymi, takimi jak pamięć taśmowa lub dyskowa. Interfejs szeregowy jest bardziej powszechny w przypadku drukarek i terminali. Wobec pojawienia się nowej generacji interfejsów szeregowych o dużej szybkości interfejsy równoległe znacznie straciły na popularności.



(a) Równoległe wejście-wyjście



(b) Szeregowe wejście-wyjście

Rysunek 7.16. Równoległe i szeregowe wejście-wyjście

W każdym przypadku moduł wejścia-wyjścia musi się angażować w dialog z urządzeniem peryferyjnym. Ogólnie rzecz biorąc, dialog w przypadku operacji zapisu wygląda następująco:

1. Moduł wejścia-wyjścia wysyła sygnał sterowania domagający się zgody na wysłanie danych.
2. Urządzenie peryferyjne potwierdza żądanie.
3. Moduł wejścia-wyjścia przesyła dane (słowo lub blok, zależnie od urządzenia peryferyjnego).
4. Urządzenie peryferyjne potwierdza otrzymanie danych.

Operacja odczytu przebiega podobnie.

Kluczem do działania modułu wejścia-wyjścia jest wewnętrzny bufor, w którym mogą być przechowywane dane przechodzące między urządzeniem peryferyjnym a resztą systemu. Bufor ten umożliwia modułowi wejścia-wyjścia kompensowanie różnic szybkości między magistralą systemową a swoimi liniami zewnętrznymi.

Konfiguracje dwupunktowe i wielopunktowe

Połączenie między modułem wejścia-wyjścia w systemie komputerowym a urządzeniami zewnętrznymi może być dwupunktowe lub wielopunktowe. W przypadku interfejsu dwupunktowego między modułem wejścia-wyjścia a urządzeniem zewnętrznym jest przewidziana specjalistyczna linia. W małych systemach (komputery osobiste, stacje robocze) typowe łącza dwupunktowe dotyczą klawiatury, drukarki i mode-

mu zewnętrznego. Typowym przykładem takiego interfejsu jest specyfikacja EIA-232 (jej opis zawarto w [STAL00]).

Wzrasta znaczenie wielopunktowych interfejsów zewnętrznych wykorzystywanych do obsługi zewnętrznych urządzeń pamięci masowej (napędów dyskowych lub taśmowych) oraz urządzeń multimedialnych (CD-ROM, wideo, audio). Interfejsy wielopunktowe są w istocie zewnętrznymi magistralami i mają takie same rodzaje rozwiązań logicznych co magistrale przedyskutowane w rozdz. 3. W tym podrozdziale omówimy dwa kluczowe przykłady: FireWire i InfiniBand.

Magistrala szeregowa FireWire

Wobec szybkości procesorów sięgających gigaherców i urządzeń pamięciowych mieszczących wiele gigabitów, wymagania odnoszące się do wejścia-wyjścia komputerów osobistych, stacji roboczych i serwerów są ogromne. Jednak technologie kanałów wejścia-wyjścia o dużej szybkości opracowane dla dużych komputerów i superkomputerów są zbyt kosztowne i rozbudowane, aby mogły znaleźć zastosowanie w tych mniejszych systemach. Dlatego pilnie poszukiwano możliwości opracowania alternatywnych, bardzo szybkich rozwiązań w stosunku do SCSI i innych interfejsów wejścia-wyjścia przeznaczonych dla małych systemów. Wynikiem tych dążeń jest standard IEEE 1394 odnoszący się do magistrali szeregowej o wysokiej wydajności, znanej powszechnie jako FireWire.

FireWire ma wiele zalet w porównaniu ze starszymi interfejsami wejścia-wyjścia. Wyróżnia się bardzo dużą szybkością, niewielkim kosztem i łatwością implementacji. W istocie FireWire znajduje zastosowanie nie tylko w systemach komputerowych, lecz również w takich produktach elektroniki konsumpcyjnej, jak cyfrowe aparaty fotograficzne, kamery wideo i telewizory. FireWire służy w nich do przesyłania obrazów wideo, które coraz częściej pochodzą ze źródeł cyfrowych.

Jedną z silnych stron interfejsu FireWire jest to, że dane są przesyłane szeregowo (bit po bicie), nie zaś równolegle. Interfejsy równoległe, takie jak SCSI, wymagają większej liczby przewodów, co oznacza szersze i kosztowniejsze kable oraz szersze i kosztowniejsze złącza o wielu szpilkach, które mogą się zgiąć lub złamać. Kabel o większej liczbie przewodów wymaga ekranowania, aby zapobiec zakłóceniom elektrycznym między przewodami. W przypadku interfejsu równoległego wymagana jest również synchronizacja między przewodami, co jest coraz bardziej kłopotliwe w miarę wydłużania się kabla.

Ponadto komputery stają się fizycznie coraz mniejsze, jeśli nawet wzrasta ich moc obliczeniowa i wymagania w odniesieniu do wejścia-wyjścia. W komputerach przenoszonych w ręku i kieszonkowych jest niewiele miejsca na złącza, a mimo to są wymagane duże szybkości przesyłania danych, aby mogły być przetwarzane obrazy i pliki wideo.

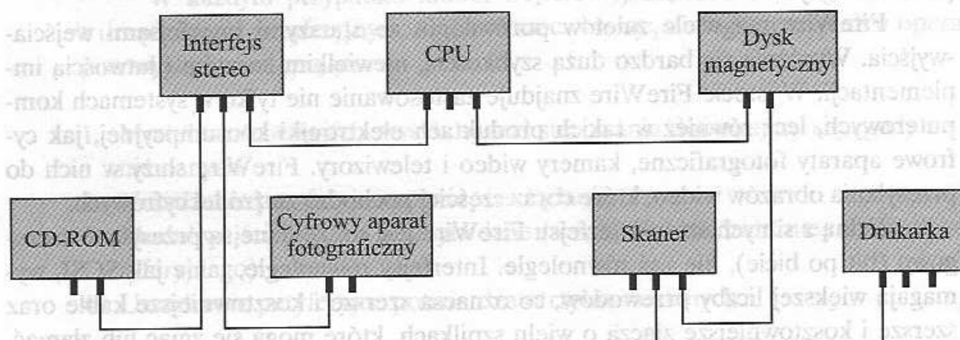
Intencją twórców FireWire było zapewnienie pojedynczego interfejsu wejścia-wyjścia z prostym złączem, który mógłby obsługiwać liczne urządzenia za pośrednictwem pojedynczego portu, dzięki czemu połączenia z myszą, drukarką laserową, zewnętrznym napędem dyskowym, źródłami dźwięku i z siecią lokalną mogłyby być

zastąpione tym właśnie, jednym złączem. Jeśli chodzi o takie złącze, inspiracja pochodziła z Nintendo Gameboy. Byłoby to tak wygodne, że użytkownik mógłby sięgnąć za komputer i wetknąć wtyczkę bez patrzenia.

Konfiguracje FireWire

W FireWire jest stosowana konfiguracja łańcuchowa (*daisy chain*), w której do jednego portu można przyłączyć do 63 urządzeń. Ponadto można ze sobą połączyć do 1022 magistral FireWire za pomocą mostków, dzięki czemu system może obsługiwać dowolną wymaganą liczbę urządzeń peryferyjnych.

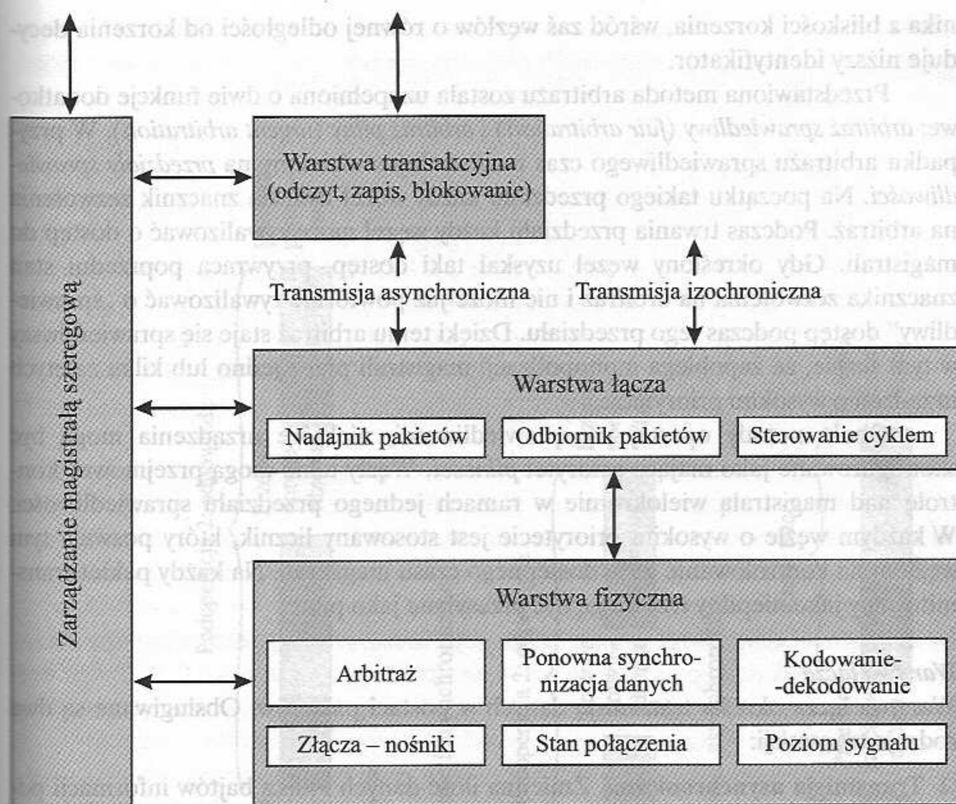
FireWire umożliwia tzw. podłączanie aktywne (*hot plugging*), a więc istnieje możliwość dołączania i odłączania urządzeń peryferyjnych bez konieczności wyłączania komputera lub ponownego konfigurowania systemu. FireWire zapewnia również automatyczną konfigurację; nie ma potrzeby ręcznego ustawiania identyfikatorów urządzeń ani zajmowania się względnym rozmieszczeniem tych urządzeń. Prosta konfiguracja FireWire została pokazana na rys. 7.17. W przypadku FireWire nie ma potrzeby stosowania specjalnych terminatorów, a system automatycznie dokonuje konfiguracji, przypisując odpowiednie adresy. Zwróćmy również uwagę, że magistrala FireWire nie musi mieć postaci ściśle łańcuchowej; jest możliwa konfiguracja drzewiasta.



Rysunek 7.17. Prosta konfiguracja FireWire

Ważną właściwością standardu FireWire jest to, że określa on zbiór trzech warstw protokołów z myślą o znormalizowaniu sposobu oddziaływania systemu macierzystego z urządzeniami peryferyjnymi poprzez magistralę szeregową. Ten stos protokołów został przedstawiony na rys. 7.18. Trzy warstwy stosu są następujące:

- **Warstwa fizyczna.** Określa nośniki transmisji dopuszczalne w FireWire oraz ich właściwości elektryczne i sygnałowe.
- **Warstwa łącza.** Opisuje pakietową transmisję danych.
- **Warstwa transakcyjna.** Określa protokół zapotrzebowanie-odpowiedź, który powoduje ukrycie szczegółów niższego rzędu FireWire przed aplikacjami.



Rysunek 7.18. Stos protokołów FireWire

Warstwa fizyczna

Warstwa fizyczna FireWire określa kilka alternatywnych nośników transmisji wraz ze złączami, różniące się własnościami fizycznymi i parametrami transmisji danych. Zdefiniowane są szybkości transmisji od 25 do 400 Mb/s. Warstwa fizyczna przekształca dane binarne na sygnały elektryczne odpowiadające różnym nośnikom fizycznym. Warstwa ta zapewnia również usługi arbitrażowe, które gwarantują, że w danej chwili tylko jedno urządzenie może transmitować dane.

FireWire zapewnia dwie formy arbitrażu. Forma najprostsza jest oparta na drzewiastym układzie węzłów na magistrali FireWire, o którym wspomniano wcześniej. Szczególnym przypadkiem takiej struktury jest liniowy układ łańcuchowy. Warstwa fizyczna zawiera układy logiczne umożliwiające samokonfigurację dołączonych urządzeń w taki sposób, że jeden węzeł jest wyznaczany jako korzeń drzewa, pozostałym zaś są przypisane zależności typu „rodzice-dzieci", dzięki czemu powstaje topologiczny układ drzewiasty. Gdy już taka konfiguracja zostanie ustanowiona, węzeł-korzeń pełni funkcję centralnego arbitra i przetwarza zapotrzebowania na dostęp do magistrali na zasadzie pierwszy zgłoszony, pierwszy obsłużony. W przypadku wystąpienia zgłoszeń równoczesnych dostęp zostaje udzielony węzłowi o najwyższym priorytecie naturalnym. Priorytet naturalny wy-

nika z bliskości korzenia, wśród zaś węzłów o równej odległości od korzenia decyduje niższy identyfikator.

Przedstawiona metoda arbitrażu została uzupełniona o dwie funkcje dodatkowe: *arbitraż sprawiedliwy* (*fair arbitration*) i *arbitraż pilny* (*urgent arbitration*). W przypadku arbitrażu sprawiedliwego czas magistrali jest dzielony na *przedziały sprawiedliwości*. Na początku takiego przedziału każdy węzeł ustawia znacznik zezwolenia na arbitraż. Podczas trwania przedziału każdy węzeł może rywalizować o dostęp do magistrali. Gdy określony węzeł uzyskał taki dostęp, przywraca poprzedni stan znacznika zezwolenia na arbitraż i nie może już powtórnie rywalizować o „sprawiedliwy" dostęp podczas tego przedziału. Dzięki temu arbitraż staje się sprawiedliwszy w tym sensie, że zapobiega monopolizacji magistrali przez jedno lub kilka zajętych urządzeń o wysokim priorytecie.

Obok metody opartej na sprawiedliwości, niektóre urządzenia mogą być skonfigurowane jako mające priorytet *pilności*. Węzły takie mogą przejmować kontrolę nad magistralą wielokrotnie w ramach jednego przedziału sprawiedliwości. W każdym węźle o wysokim priorytecie jest stosowany licznik, który pozwala tym węzłom na kontrolowanie 75% dostępnego czasu magistrali. Na każdy pakiet transmitowany jako niepilny trzy mogą być przesyłane jako pilne.

Warstwa łącza

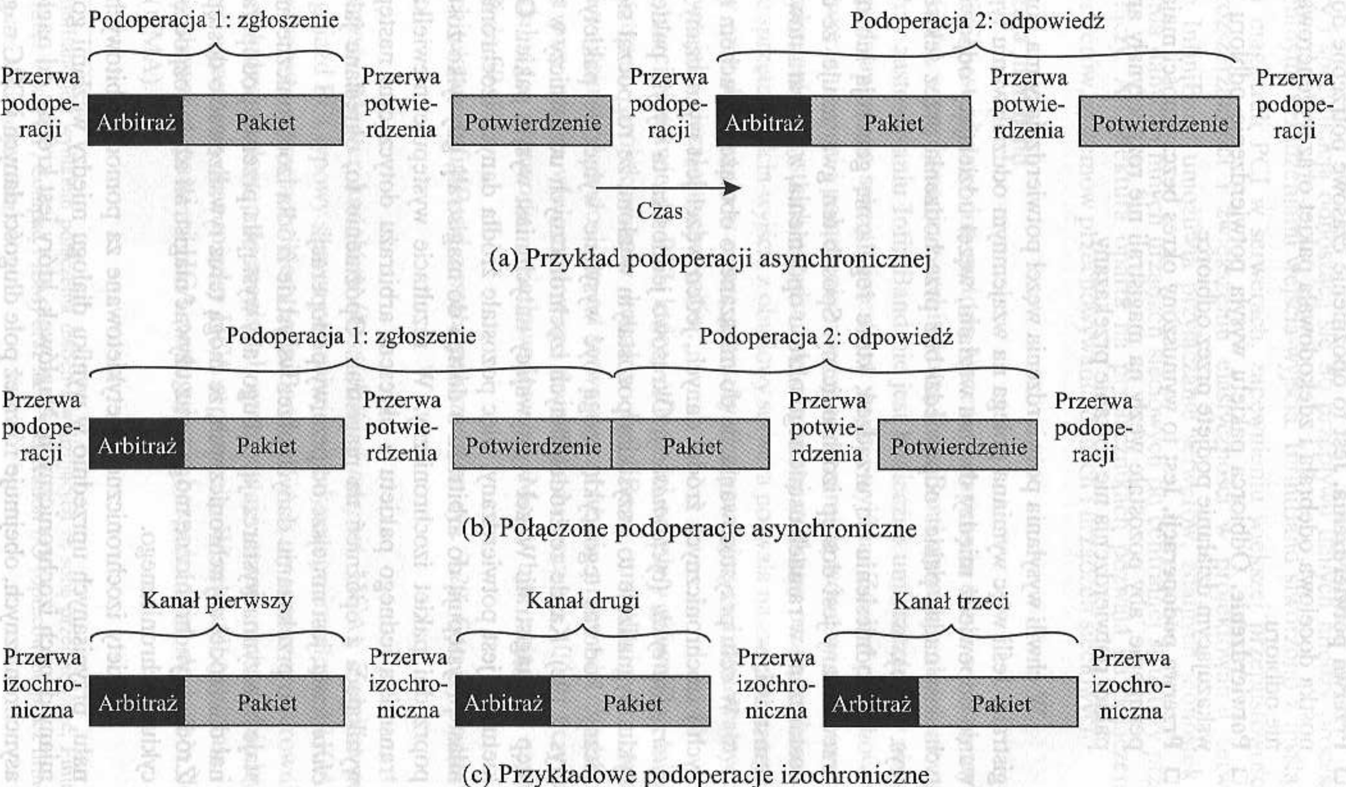
Warstwa łącza określa transmisję danych w postaci pakietów. Obsługiwane są dwa rodzaje transmisji:

- **Transmisja asynchroniczna.** Zmienna ilość danych i kilka bajtów informacji pochodzących z warstwy transakcyjnej jest transmitowanych jako pakiet pod określony adres, po czym powraca potwierdzenie odbioru.
- **Transmisja izochroniczna.** Zmienna ilość danych jest przesyłana w postaci szeregu pakietów o ustalonym rozmiarze w regularnych odstępach czasowych. W takiej postaci transmisji adresowanie jest uproszczone i nie stosuje się potwierdzeń.

Transmisja asynchroniczna jest stosowana w odniesieniu do danych nie wymagających ustalonej szybkości przesyłania. W przypadku transmisji synchronicznej może być stosowany zarówno arbitraż sprawiedliwy, jak i pilny. Metodą domyślną jest arbitraż sprawiedliwy. Wobec urządzeń wymagających znacznej części przepustowości magistrali lub mających szczególne wymagania, jeśli chodzi o opóźnienia, jest stosowana metoda arbitrażu pilnego. Na przykład węzeł gromadzenia danych w czasie rzeczywistym o dużej szybkości może korzystać z arbitrażu pilnego, gdy wypełnienie buforów danych o znaczeniu krytycznym przekracza połowę.

Na rysunku 7.19a przedstawiono typową transakcję asynchroniczną. Proces dostarczania pojedynczego pakietu nosi nazwę podoperacji (*subaction*). Składa się on z pięciu okresów. Są to:

- **Sekwencja arbitrażu.** Jest to wymiana sygnałów wymagana do tego, aby jedno z urządzeń uzyskało kontrolę nad magistralą.
- **Transmisja pakietowa.** Każdy pakiet ma nagłówek z identyfikatorami źródła i miejscem docelowym. Nagłówek zawiera również informację o rodzaju pakietu,



Rysunek 7.19. Podoperacje FireWire

sumę kontrolną cyklicznej kontroli nadmiarowej (*cyclic redundancy check* – CRC) i informację o parametrach określonego rodzaju pakietów. Pakiet może również zawierać blok danych składający się z danych użytkownika i kolejnej sumy CRC.

- **Przerwa potwierdzenia.** Jest to opóźnienie czasowe potrzebne do tego, aby jednostka docelowa odebrała i zdekodowała pakiet oraz wygenerowała potwierdzenie odbioru.
- **Potwierdzenie.** Odbiorca pakietu wysyła potwierdzenie odbioru wraz z kodem wskazującym działanie podjęte przez odbiorcę.
- **Przerwa podoperacji.** Jest to wymuszony okres bezczynności mający na celu zapewnienie, aby pozostałe węzły na magistrali nie rozpoczynały arbitrażu, zanim pakiet potwierdzenia nie zostanie przekazany.

W chwili wysyłania potwierdzenia węzeł potwierdzający ma kontrolę nad magistralą. Jeśli więc wymiana polega na wzajemnym oddziaływaniu typu zapotrzebowanie-odpowiedź między dwoma węzłami, węzeł udzielający odpowiedzi może natychmiast nadać pakiet odpowiedzi bez przechodzenia przez sekwencję arbitrażu (rys. 7.19b).

W odniesieniu do urządzeń, które regularnie generują lub odbierają dane, przewidziany jest dostęp izochroniczny. Sposób ten gwarantuje, że dane mogą być dostarczane w ramach ustalonego okresu opóźnienia, z zagwarantowaną szybkością transmisji.

W celu przystosowania się do mieszanego obciążenia ruchem z asynchronicznych i izochronicznych źródeł danych, jeden węzeł jest wyznaczany jako *jednostka nadrzędna cyklu* (*cycle master*). Okresowo jednostka ta wysyła pakiety rozpoczęcia cyklu. Sygnalizuje to wszystkim pozostałym węzłom, że rozpoczął się cykl izochroniczny. Podczas tego cyklu mogą być wysyłane wyłącznie pakiety izochroniczne (rys. 7.19c). Każde ze źródeł danych izochronicznych uczestniczy w arbitrażu o dostęp do magistrali. Węzeł wygrywający natychmiast wysyła pakiet. Odbiór tego pakietu nie jest potwierdzany, więc pozostałe źródła danych izochronicznych natychmiast przystępują do arbitrażu o dostęp do magistrali, gdy tylko zostanie przesłany poprzedni pakiet izochroniczny. W rezultacie występuje niewielka luka między transmisją jednego pakietu a okresem arbitrażu dotyczącym następnego pakietu, wynikająca z opóźnień na magistrali. Opóźnienie to, określane jako *przerwa izochroniczna*, jest mniejsze od przerwy podoperacji.

Po przekazaniu danych przez wszystkie źródła izochroniczne magistrala pozostaje bezczynna wystarczająco długo, aby wystąpiła przerwa podoperacji. Jest to sygnał dla źródeł asynchronicznych, że mogą teraz rywalizować o dostęp do magistrali. Źródła asynchroniczne mogą teraz używać magistrali aż do rozpoczęcia następnego cyklu izochronicznego.

Pakiety izochroniczne są etykietowane za pomocą 8-bitowych numerów kanału, przypisanych uprzednio w wyniku dialogu między węzłami gotowymi do wymiany danych izochronicznych. Nagłówek, który jest krótszy od nagłówka pakietów asynchronicznych, obejmuje również pole długości danych i CRC nagłówka.

InfiniBand

InfiniBand to niedawno opracowana specyfikacja wejścia-wyjścia ukierunkowana na rynek serwerów o najwyższej jakości[4]. Pierwsza wersja tej specyfikacji ukazała się na początku roku 2001 i przyciągnęła wielu producentów. Standard ten jest opisem architektury i specyfikacją odnoszącą się do przepływu danych między procesorami a inteligentnymi urządzeniami wejścia-wyjścia. InfiniBand stworzono z myślą o zastąpieniu magistrali PCI w serwerach, zapewnieniu większej przepustowości, poprawie rozszerzalności i zwiększeniu elastyczności w projektowaniu serwerów. W istocie InfiniBand umożliwia przyłączanie serwerów, zdalnych jednostek przechowywania danych i innych urządzeń sieciowych do centralnej struktury przełączników i łączy. Architektura oparta na przełącznikach pozwala na połączenie do 64 000 serwerów, systemów przechowywania danych i urządzeń sieciowych.

Architektura InfiniBand

Chociaż PCI jest niezawodną metodą łączenia i zapewnia coraz większą szybkość (aż do 1 Gb/s), w porównaniu z architekturą InfiniBand wykazuje określone ograniczenia. Przy zastosowaniu InfiniBand nie jest konieczne umieszczanie w obudowie serwera podstawowego sprzętu interfejsu wejścia-wyjścia. Zdalne przechowywanie danych, tworzenie sieci i połączenia między serwerami są wówczas realizowane poprzez przyłączenie wszystkich urządzeń do centralnej struktury przełączników i łączy. Usunięcie sprzętu wejścia-wyjścia z obudowy serwera pozwala na zwiększenie gęstości zabudowy oraz dodawanie w miarę potrzeby elastyczniejszych i skalowalnych centrów danych jako niezależnych węzłów.

W przeciwieństwie do PCI, której odległości od płyty głównej są mierzone w centymetrach, projektowanie kanałów InfiniBand umożliwia umieszczanie urządzeń wejścia-wyjścia do 17 m od serwera przy użyciu przewodu miedzianego, do 300 m przy użyciu wielomodowego światłowodu oraz do 10 km przy użyciu jednomodalnego przewodu światłowodowego. Mogą być osiągane szybkości transmisji do 30 Gb/s.

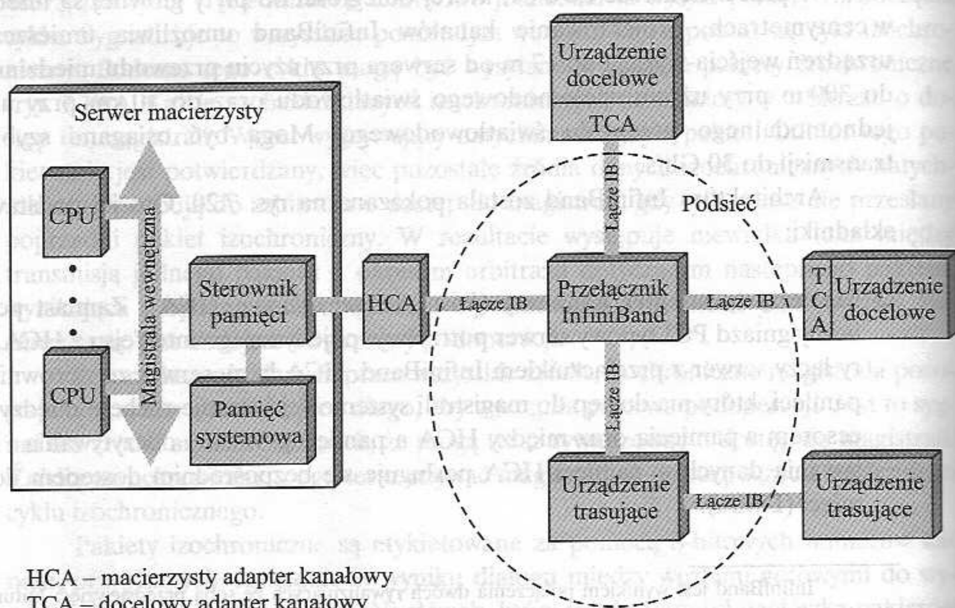
Architektura InfiniBand została pokazana na rys. 7.20. Oto jej podstawowe składniki:

- **Macierzysty adapter kanałowy** (*host channel adapter*, HCA). Zamiast pewnej liczby gniazd PCI typowy serwer potrzebuje pojedynczego interfejsu z HCA, który łączy serwer z przełącznikiem InfiniBand. HCA łączy serwer ze sterownikiem pamięci, który ma dostęp do magistrali systemowej i steruje ruchem między procesorem a pamięcią oraz między HCA a pamięcią. W celu odczytywania i zapisywania danych w pamięci HCA posługuje się bezpośrednim dostępem do pamięci (DMA).

[4] InfiniBand jest wynikiem połączenia dwóch rywalizujących ze sobą przedsięwzięć: Future I/O (popierane przez Cisco, HP, Compaq i IBM) oraz Next Generation I/O (opracowane przez firmę Intel i popierane przez kilka innych firm).

- **Docelowy adapter kanałowy** (*target channel adapter* – TCA). TCA służy do łączenia systemów przechowywania danych, urządzeń trasujących i innych urządzeń peryferyjnych z przełącznikiem InfiniBand.
- **Przełącznik InfiniBand.** Przełącznik ten zapewnia fizyczne połączenia dwupunktowe z różnorodnymi urządzeniami i przełącza ruch z jednego łącza do drugiego. Serwery i urządzenia komunikują się poprzez swoje adaptery, za pośrednictwem przełącznika. „Inteligentne" układy przełącznika zarządzają połączeniami bez zakłócania pracy serwerów.
- **Łącza.** Łącze między przełącznikiem a adapterem kanału lub między dwoma przełącznikami.
- **Podsieć.** Podsieć składa się z jednego lub z wielu wzajemnie połączonych przełączników oraz z łączy między innymi urządzeniami a tymi przełącznikami. Na rysunku 7.20 została pokazana podsieć z jednym przełącznikiem; gdy jednak ma być połączona wielka liczba urządzeń, wymagane są bardziej złożone podsieci. Podsieci umożliwiają administratorom ograniczanie transmisji rozgłoszeniowych i grupowych w ramach podsieci.
- **Urządzenie trasujące.** Łączy podsieci InfiniBand lub łączy przełącznik InfiniBand z siecią, taką jak sieć lokalna, sieć rozległa lub sieć obszaru przechowywania danych.

Adaptery kanałowe są inteligentnymi urządzeniami, które realizują wszystkie funkcje wejścia-wyjścia bez potrzeby zakłócania procesora serwera. Istnieje na przykład protokół sterowania, za pomocą którego przełącznik wykrywa wszystkie TCA i HCA w danej strukturze i przypisuje każdemu z nich adresy logiczne. Jest to czynione bez angażowania procesora.



Rysunek 7.20. Struktura przełączników InfiniBand

Przełącznik InfiniBand czasowo otwiera kanały między procesorem (CPU) a urządzeniami, z którymi się on komunikuje. Urządzenia nie muszą dzielić ze sobą przepustowości kanału, jak w przypadku rozwiązań opartych na magistrali (takich jak PCI), w których urządzenia muszą się poddawać arbitrażowi w celu uzyskania dostępu do procesora. Dodatkowe urządzenia wprowadza się do tej konfiguracji poprzez łączenie ich TCA z przełącznikiem.

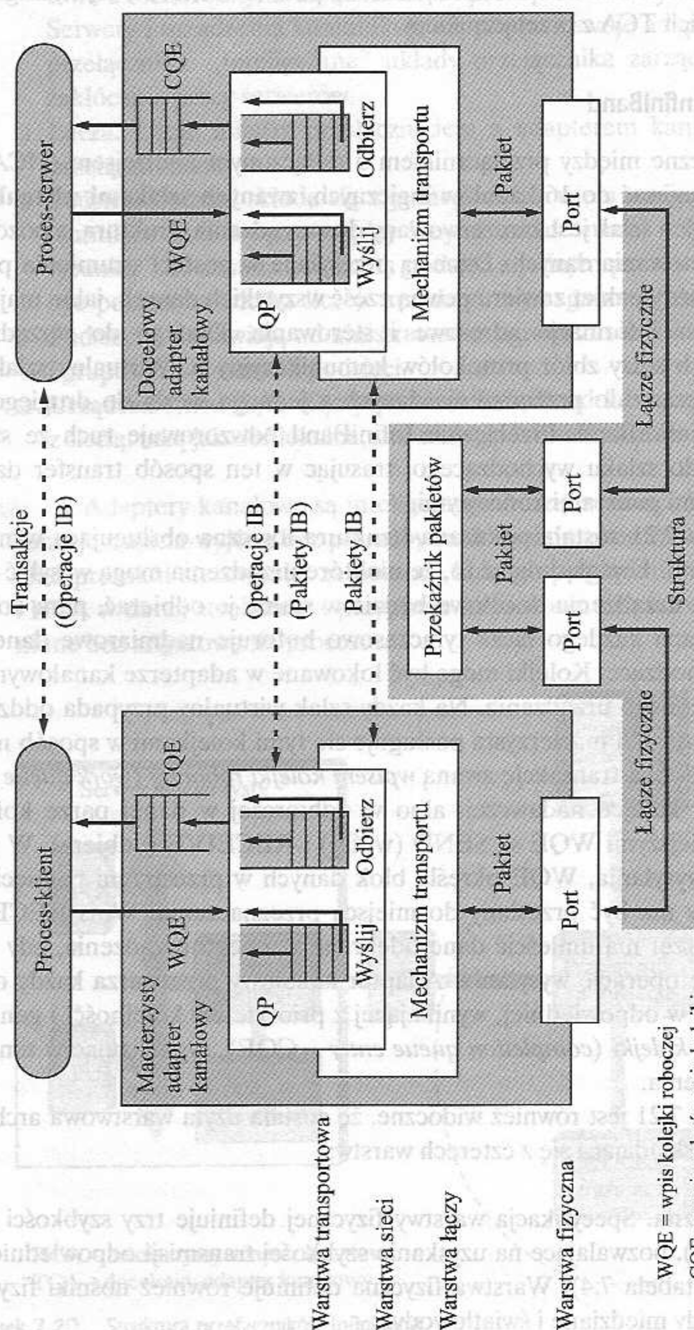
Funkcjonowanie InfiniBand

Każde łącze fizyczne między przełącznikiem a dołączonym interfejsem (HCA lub TCA) może obsługiwać do 16 kanałów logicznych, zwanych **szlakami wirtualnymi** (*virtual lanes*). Jeden szlak jest zarezerwowany do zarządzania strukturą, a pozostałe służą do transportowania danych. Dane są przesyłane w postaci strumienia pakietów, przy czym każdy pakiet zawiera pewną część wszystkich danych, jakie mają być przeniesione, oraz informacje adresowe i sterowania. Dlatego do zarządzania transferem danych służy zbiór protokołów komunikacyjnych. Wirtualny szlak jest czasowo przeznaczany do przenoszenia danych z jednego węzła do drugiego poprzez strukturę InfiniBand. Przełącznik InfiniBand odwzorowuje ruch ze szlaku przychodzącego do szlaku wychodzącego, trasując w ten sposób transfer danych między pożądanymi punktami końcowymi.

Na rysunku 7.21 została pokazana struktura logiczna obsługująca wymianę poprzez InfiniBand. Uwzględniając to, że niektóre urządzenia mogą wysyłać dane szybciej, niż inne urządzenie docelowe będzie w stanie je odbierać, para kolejek na obydwu końcach każdego łącza tymczasowo buforuje nadmiarowe dane wychodzące i przychodzące. Kolejki mogą być lokowane w adapterze kanałowym lub w pamięci dołączonego urządzenia. Na każdy szlak wirtualny przypada oddzielna para kolejek. Jednostka macierzysta posługuje się tymi kolejkami w sposób następujący. Umieszcza ona transakcję zwaną *wpisem kolejki roboczej* (*work queue entry* – WQE) albo w kolejce nadawczej, albo w odbiorczej w danej parze kolejek. Dwoma najważniejszymi WQE są SEND (wyślij) i RECEIVE (odbierz). W przypadku operacji wysyłania, WQE określa blok danych w przestrzeni pamięciowej urządzenia, który ma być przesłany do miejsca przeznaczenia. Wpis RECEIVE określa, gdzie sprzęt ma umieścić dane odebrane z innego urządzenia, gdy użytkownik wykonuje operację wysyłania. Adapter kanałowy przetwarza każdy otrzymany wpis WQE w odpowiedniej, wynikającej z priorytetów kolejności i generuje *wpis zakończenia kolejki* (*completion queue entry* – CQE), sygnalizując w ten sposób stan zakończenia.

Na rysunku 7.21 jest również widoczne, że została użyta warstwowa architektura protokołów, składająca się z czterech warstw:

- **Warstwa fizyczna.** Specyfikacja warstwy fizycznej definiuje trzy szybkości łączy (1X, 4X i 12X), pozwalające na uzyskanie szybkości transmisji odpowiednio 2,5, 10 i 30 Gb/s (tabela 7.4). Warstwa fizyczna definiuje również nośniki fizyczne, w tym przewody miedziane i światłowody.



Rysunek 7.21. Stos protokołów komunikacyjnych InfiniBand

- **Warstwa łączy.** Ta warstwa określa podstawową strukturę pakietów służących do wymiany danych, w tym system adresowania przypisujący unikalny adres łącza każdemu urządzeniu w podsieci. Poziom ten zawiera układy logiczne służące do konfigurowania wirtualnych szlaków i do przełączania danych poprzez przełączniki od źródła do miejsca przeznaczenia w ramach podsieci. W celu zapewnienia niezawodności struktura pakietów obejmuje kod wykrywania błędów.
- **Warstwa sieci.** Warstwa sieci trasuje pakiety między różnymi podsieciami InfiniBand.
- **Warstwa transportowa.** Warstwa transportowa zapewnia niezawodność transferu pakietów na całej ich trasie, poprzez jedną lub wiele podsieci.

Tabela 7.4. Przepustowość łączy i danych InfiniBand

Łącze	Szybkość sygnałów (jednokierunkowa)	Przepustowość używalna (80% szybkości sygnałów)	Efektywna szybkość transferu danych (wysłanie + odebranie)
O szerokości pojedynczej	2,5 Gb/s	2 Gb/s (250 MB/s)	(250 + 250) MB/s
O szerokości poczwórnej	10 Gb/s	8 Gb/s (1 GB/s)	(1 + 1) GB/s
O szerokości 12-krotnej	30 Gb/s	24 Gb/s (3 GB/s)	(3 + 3) GB/s

7.8. Polecana literatura i witryny WWW

Dobre omówienie architektury modułów wejścia-wyjścia firmy Intel, łącznie z 82C59A i 82C55A, można znaleźć w [BREY00].

FireWire przedstawiono szczegółowo w [ANDE98]. W [WICK97] i [THOM00] zawarto zwarte przeglądy dotyczące FireWire.

InfiniBand przedstawiono szczegółowo w [FUTR01]. Zwarty przegląd znajduje się w [KAGA01].

ANDE98 Anderson D.: *FireWire System Architecture.* Reading, Addison-Wesley, 1998.

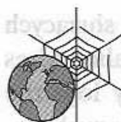
BREY00 Brey B.: *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro and Pentium II Processors.* Upper Saddle River, Prentice Hall, 2000.

FUTR01 Futral W.: *InfiniBand Architecture: Development and Deployment.* Hillsboro, Intel Press, 2001.

KAGA01 Kagan M.: „InfiniBand: Thinking Outsider the Box Design". *Communications System Design,* September 2001 (www.csdmag.com).

THOM00 Thompson D.: „IEEE 1394: Changing the Way We Do Multimedia Communications". *IEEE Multimedia,* April-June 2000.

WICK97 Wickelgren I.: „The Facts Abort FireWire". *IEEE Spectrum,* April 1997.



Polecane witryny WWW:

- **T10 Home Page.** T10 to Technical Committee of the National Committee on Information Technology Standards (Komitet Techniczny Narodowego Komitetu ds. Standardów Technologii Informacyjnej). Jest odpowiedzialny za interfejsy niskiego poziomu. Jego głównym dziełem jest Small Computer System Interface (interfejs małych systemów komputerowych, SCSI).
- **1394 Trade Association.** Zawiera informacje techniczne i łącza do producentów FireWire.
- **InfiniBand Trade Association.** Zawiera informacje techniczne i łącza do producentów InfiniBand.

7.9. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Bezpośredni dostęp do pamięci (DMA) – *direct memory access*
FireWire
InfiniBand
Izolowane wejście-wyjście – *isolated I/O*
Kanał multiplekserowy – *multiplexor channel*
Kanał wejścia-wyjścia – *I/O chanel*
Kanał wybiórczy – *selektor channel*
Kradzenie cyklu – *cycle stealing*
Moduł wejścia-wyjścia – *I/O module*
Polecenie wejścia-wyjścia – *I/O command*
Procesor wejścia-wyjścia – *I/O procesor*
Programowane wejście-wyjście – *programmed I/O*
Przerwanie – *interrupt*
Równoległe wejście-wyjście – *parallel I/O*
Szeregowe wejście-wyjście – *serial I/O*
Urządzenie peryferyjne – *peripheral device*
Wejście-wyjście odwzorowane w pamięci – *memory-mapped I/O*
Wejście-wyjście sterowane przerwaniami – *interrupt-driven I/O*

Pytania kontrolne

7.1. Wymień trzy ogólne klasyfikacje urządzeń zewnętrznych (lub peryferyjnych).
7.2. Co to jest Międzynarodowy Alfabet Wzorcowy?
7.3. Jakie są główne funkcje modułu wejścia-wyjścia?
7.4. Wymień i krótko zdefiniuj trzy techniki realizowania operacji wejścia-wyjścia.
7.5. Jaka jest różnica między wejściem-wyjściem odwzorowanym w pamięci a izolowanym?
7.6. Gdy nastąpiło przerwanie ze strony urządzenia, w jaki sposób procesor określa, które urządzenie wysłało to przerwanie?
7.7. Gdy moduł DMA przejmuje kontrolę nad magistralą i ją zachowuje, co wówczas robi procesor?

Problemy do rozwiązania

7.1. W podrozdziale 7.3 wymieniliśmy jedną wadę i jedną zaletę wejścia-wyjścia odwzorowanego w pamięci w porównaniu z izolowanym. Wymień dwie inne wady i dwie inne zalety.

7.2. Praktycznie we wszystkich systemach zawierających moduły DMA dostęp DMA do pamięci głównej ma wyższy priorytet niż dostęp procesora do tej pamięci. Dlaczego tak jest?

7.3. Powtórz problem 7.4, stosując DMA i zakładając jedno przerwanie na sektor.

7.4. Moduł DMA przesyła do pamięci znaki z urządzenia transmitującego z szybkością 9600 bit/s, stosując wykradanie cyklu. Procesor pobiera rozkazy z szybkością miliona rozkazów na sekundę. O ile procesor będzie spowolniony przez moduł DMA?

7.5. Komputer 32-bitowy ma dwa kanały wybiórcze i jeden multiplekserowy. Każdy kanał wybiórczy obsługuje dwie jednostki dysków magnetycznych i dwie jednostki taśmowe. Kanał multiplekserowy obsługuje zaś dwie drukarki wierszowe, dwa czytniki kart i dziesięć terminali wizyjnych. Załóżmy następujące szybkości przesyłania:

napęd dysków	800 KB/s
napęd taśmowy	200 KB/s
drukarka wierszowa	6,6 KB/s
czytnik kart	1,2 KB/s
terminal wizyjny	1 KB/s.

Oszacuj maksymalną łączną szybkość przesyłania w tym systemie.

7.6. Komputer składa się z procesora i urządzenia wejścia-wyjścia D połączonego z pamięcią główną za pomocą wspólnej magistrali o szerokości 1 słowa. Procesor może wykonywać maksymalnie 10^6 rozkazów na sekundę. Przeciętny rozkaz wymaga 5 cykli maszynowych, z których 3 używają magistrali pamięci. Operacja odczytu lub zapisu pamięci zajmuje 1 cykl maszynowy. Załóżmy, że procesor w sposób ciągły realizuje programy podstawowe wykorzystujące 95% jego zdolności wykonywania rozkazów, jednak bez rozkazów wejścia-wyjścia. Przyjmijmy, że czas cyklu maszynowego jest równy czasowi cyklu magistrali. Załóżmy teraz, że urządzenie wejścia-wyjścia ma być użyte do przesłania bardzo dużych bloków danych z (i do) pamięci głównej.

(a) Jeśli stosowane jest programowane wejście-wyjście, a przesłanie każdego słowa wymaga od procesora wykonania 2 rozkazów, oszacuj maksymalną szybkość przesyłania danych z wejścia i na wyjście w słowach na sekundę, możliwą do uzyskania przez D.

(b) Oszacuj tę samą szybkość, jeśli używane jest DMA.

7.7. Źródło danych dostarcza 7-bitowych znaków IRA, przy czym do każdego z nich jest dołączony bit parzystości. Wyprowadź wzór na maksymalną efektywną szybkość transferu danych (bitów danych IRA) przez linię R (w bitach na sekundę) dla następujących przypadków:

(a) transmisja asynchroniczna z bitem stopu 1,5 jednostki;

(b) transmisja bitowo-synchroniczna, przy czym porcja danych zawiera 48 bitów sterowania i 128 bitów danych;

(c) to samo co w (b), jednak pole informacyjne obejmuje 1024 bity;

(d) transmisja znakowo-synchroniczna z 9 znakami sterowania i 16 znakami informacji na porcję;

e) to samo co w (d), jednak ze 128 znakami na porcję.

7.8. Następujący problem wykorzystuje ilustrację mechanizmu wejścia-wyjścia zawartą w [ECKE90] (rys. 7.22):

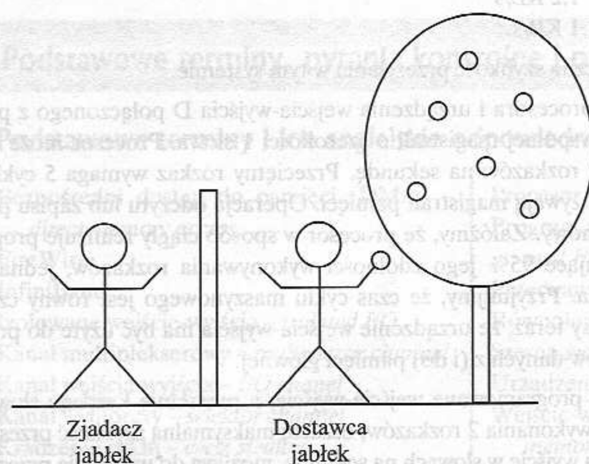
Dwaj chłopcy bawią się po obu stronach wysokiego ogrodzenia. Jeden z nich, nazwany Dostawcą jabłek, dysponuje piękną jabłonką obsypaną smacznymi jabłkami, rosnącą po jego stronie ogrodzenia. Jest on szczęśliwy, mogąc podawać jabłka drugiemu chłopcu, gdy tylko ten ich potrzebuje. Drugi chłopiec, nazwany Zjadaczem jabłek, lubi jeść jabłka, jed-

nak nie ma żadnego. W rzeczywistości, może on jeść jabłka w stałym tempie („jedno jabłko dziennie, a unikniesz wizyty u lekarza"). Jeśli jadłby jabłka szybciej, zachorowałby. Jeśli jadłby je wolniej, byłby niedożywiony. Żaden z chłopców nie może mówić, tak więc problem polega na dostarczaniu jabłek Zjadaczowi przez Dostawcę z właściwą szybkością.

(a) Załóżmy, że na szczycie ogrodzenia znajduje się budzik umożliwiający wielokrotny alarm. Jak można użyć budzika do rozwiązania tego problemu? Narysuj wykres przebiegów czasowych ilustrujący to rozwiązanie.

(b) Załóżmy teraz, że nie ma budzika. Zamiast niego Zjadacz ma flagę, którą powiewa, gdy potrzebuje jabłka. Zasugeruj nowe rozwiązanie. Czy byłoby pomocne, gdyby Dostawca również dysponował flagą? Jeśli tak, to uwzględnij to w rozwiązaniu. Przedyskutuj wady tego rozwiązania.

(c) Usuńmy teraz flagę i załóżmy istnienie długiego sznurka. Zasugeruj rozwiązanie wykorzystujące sznurek, które byłoby lepsze od (b).



Rysunek 7.22. Problem jabłek

7.9. Załóżmy, że jeden 16-bitowy i dwa 8-bitowe mikroprocesory są dołączone do magistrali systemowej. Określone są następujące szczegóły:

1. Wszystkie mikroprocesory mają rozwiązania sprzętowe konieczne do wykonania dowolnego rodzaju transferu danych (programowane wejście-wyjście, wejście-wyjście sterowane przerwaniami i DMA).
2. Wszystkie mikroprocesory mają 16-bitową szynę adresową.
3. Dwa moduły pamięci, każdy o pojemności 64 KB, są dołączone do magistrali. Projektant chce użyć pamięci wspólnej tak dużej, jak tylko jest to możliwe.
4. Magistrala systemowa obsługuje maksymalnie 4 linie przerwania i 1 linię DMA.

Dokonaj innych niezbędnych założeń, po czym:

(a) Podaj specyfikację magistrali systemowej wyrażoną liczbą i rodzajem linii.

(b) Opisz możliwy protokół komunikacji na magistrali, tzn. sekwencje odczytu i zapisu, przerwania i DMA.

(c) Wyjaśnij, w jaki sposób te urządzenia są podłączone do magistrali systemowej.

Źródło: [ALEX93].

Rozdział 8

Wspieranie systemu operacyjnego

8.1. Przegląd systemów operacyjnych
Cele i funkcje systemu operacyjnego
Rodzaje systemów operacyjnych
8.2. Szeregowanie
Szeregowanie długookresowe
Szeregowanie średniookresowe
Szeregowanie krótkookresowe
8.3. Zarządzanie pamięcią
Wymiana
Partycjonowanie
Stronicowanie
Pamięć wirtualna
Bufor translacji adresów tablic stron
Segmentacja
8.4. Zarządzanie pamięcią w Pentium II i PowerPC
Rozwiązanie sprzętowe zarządzania pamięcią w Pentium II
Rozwiązanie sprzętowe zarządzania pamięcią w PowerPC
8.5. Polecana literatura i witryny WWW
8.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

Podstawowe spostrzeżenia

- System operacyjny (*operating system* – OS) jest oprogramowaniem, które steruje wykonywaniem programów przez procesor i zarządza jego zasobami. Wiele funkcji wykonywanych przez OS, łącznie z procesem szeregowania i zarządzania pamięcią, może być realizowanych skutecznie i szybko tylko wówczas, gdy rozwiązania sprzętowe procesora obejmują zdolność do wspierania OS. Praktycznie wszystkie procesory w mniejszym lub większym stopniu taką zdolnością dysponują, w tym rozwiązaniami sprzętowymi zarządzania pamięcią wirtualną i zarządzania procesami. Do tych rozwiązań należą specjalne rejestry i bufory, a także układy realizujące zadania zarządzania podstawowymi zasobami.
- Jedną z najważniejszych funkcji OS jest szeregowanie procesów (zadań). OS określa, który proces powinien być realizowany w danej chwili. Zwykle od czasu do czasu sprzęt przerwie bieżący proces, aby umożliwić systemowi operacyjnemu podjęcie nowych decyzji dotyczących szeregowania w sposób zapewniający właściwy podział czasu procesora między różne procesy.
- Kolejną ważną funkcją OS jest zarządzanie pamięcią. Najnowsze systemy operacyjne umożliwiają tworzenie pamięci wirtualnej, z czego wynikają dwie korzyści: (1) określony proces może być realizowany w pamięci głównej bez konieczności jednoczesnej obecności wszystkich rozkazów i danych tego programu w tejże pamięci; (2) łączna przestrzeń pamięci dostępna dla danego programu może przekraczać rzeczywistą pojemność pamięci głównej systemu. Chociaż zarządzanie pamięcią jest realizowane programowo, OS jest uzależniony od sprzętowego wsparcia ze strony procesora, w tym od sprzętowych rozwiązań stronicowania i segmentacji.

Chociaż tematem tej książki jest sprzęt komputerowy, istnieje dziedzina oprogramowania, która musi być omówiona: system operacyjny komputera. System operacyjny jest programem, który zarządza zasobami komputera, obsługuje programistów i szereguje wykonywanie innych programów. Zrozumienie działania systemów operacyjnych jest ważne dla wyjaśnienia mechanizmów, za pomocą których procesor steruje systemem komputerowym. W szczególności w tym właśnie kontekście można najlepiej wyjaśnić wpływ przerwań oraz zarządzanie hierarchią pamięci.

Rozdział ten rozpoczyna się od ogólnego przeglądu i krótkiej historii systemów operacyjnych. Główna część rozdziału jest poświęcona dwóm funkcjom systemu operacyjnego, które są najważniejsze przy analizowaniu organizacji i architektury komputerów: szeregowaniu i zarządzaniu pamięcią.

8.1. Przegląd systemów operacyjnych

Cele i funkcje systemu operacyjnego

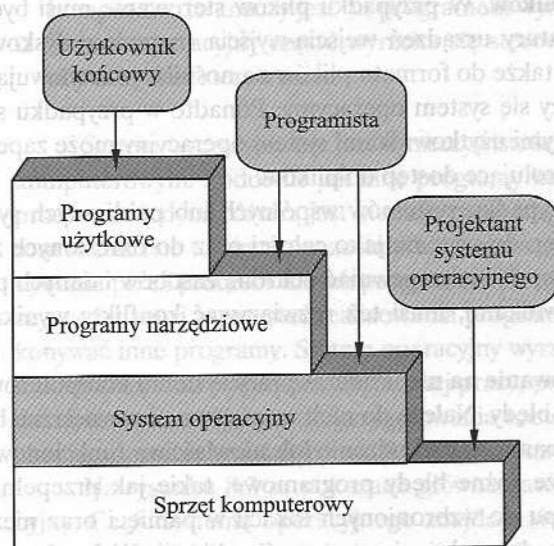
System operacyjny jest programem, który steruje wykonywaniem programów użytkowych i działa jako interfejs między użytkownikiem a sprzętem komputerowym. Można uważać, że system operacyjny ma dwa cele lub realizuje dwie funkcje:

- *Wygoda*. System operacyjny czyni system komputerowy wygodniejszym do użytku.
- *Sprawność*. System operacyjny umożliwia sprawne eksploatowanie zasobów systemu komputerowego.

Przeanalizujmy kolejno te dwa aspekty systemu operacyjnego.

System operacyjny jako interfejs użytkownik-komputer

Sprzęt i oprogramowanie umożliwiające użytkownikowi posługiwanie się aplikacjami mogą być postrzegane jako struktura warstwowa lub hierarchiczna, co zostało pokazane na rys. 8.1. Użytkownik tych aplikacji – *użytkownik końcowy* – na ogół nie interesuje się architekturą komputerów. Patrzy on na system komputerowy poprzez swój program użytkowy. Program ten może być wyrażony w języku programowania i jest przygotowywany przez programistę tworzącego oprogramowanie użytkowe. Jeśli ktoś miałby opracować program użytkowy jako zestaw rozkazów maszynowych w pełni odpowiedzialnych za sterowanie sprzętem komputerowym, stanąłby przed przytłaczająco złożonym zadaniem. Dla ułatwienia tego zadania przewidziano zestaw programów



Rysunek 8.1. Warstwy i punkty postrzegania systemu komputerowego

systemowych. Niektóre spośród tych programów są określane jako **programy narzędziowe**. Umożliwiają one realizowanie często używanych funkcji, które wspomagają tworzenie programów, zarządzanie plikami oraz sterowanie urządzeniami wejścia-wyjścia. Programista wykorzystuje te ułatwienia, przygotowując program użytkowy, a program ten podczas pracy uruchamia programy narzędziowe w celu wykonania pewnych funkcji. Najważniejszym programem systemowym jest **system operacyjny**. System operacyjny maskuje przed programistą szczegóły sprzętowe i dostarcza mu wygodnego interfejsu z systemem komputerowym. Działa jako pośrednik, ułatwiając programiście i programom użytkowym dostęp do tych udogodnień i usług.

Mówiąc w skrócie, system operacyjny zapewnia zwykle usługi należące do następujących obszarów:

- **Tworzenie programów.** System operacyjny dostarcza wielu ułatwień i usług, takich jak edytory tekstu i programy służące do usuwania błędów, wspomagających programistę przy tworzeniu programów. Zwykle usługi te mają postać programów narzędziowych (*utilities*), które w rzeczywistości nie stanowią części systemu operacyjnego, lecz są dostępne poprzez system operacyjny.
- **Wykonywanie programów.** Aby program był wykonany, musi być zrealizowanych wiele zadań. Rozkazy i dane muszą być załadowane do pamięci głównej, urządzenia wejścia-wyjścia i pliki muszą być zainicjowane, potrzebne jest też przygotowanie pozostałych zasobów. Tym wszystkim za użytkownika zajmuje się system operacyjny.
- **Dostęp do urządzeń wejścia-wyjścia.** Każde urządzenie wejścia-wyjścia do działania wymaga własnego, specyficznego zestawu rozkazów lub sygnałów sterowania. System operacyjny zajmuje się tymi szczegółami, dzięki czemu programista może myśleć w kategoriach prostych odczytów i zapisów.
- **Kontrolowany dostęp do plików.** W przypadku plików sterowanie musi być dostosowane nie tylko do natury urządzeń wejścia-wyjścia (napędów dyskowych, napędów taśmowych), lecz także do formatu plików na nośniku przechowującym. I znów, o szczegóły troszczy się system operacyjny. Ponadto w przypadku systemów z wieloma jednoczesnymi użytkownikami system operacyjny może zapewnić mechanizmy ochrony, kontrolujące dostęp do plików.
- **Dostęp do systemu.** W przypadku systemów wspólnych lub publicznych system operacyjny kontroluje dostęp do systemu jako całości oraz do określonych zasobów systemu. Funkcja dostępu musi zapewniać ochronę zasobów i danych przed nieupoważnionymi użytkownikami, musi też rozwiązywać konflikty wynikające z ograniczenia zasobów.
- **Wykrywanie błędów i reagowanie na nie.** Podczas pracy systemu komputerowego mogą wystąpić różnorodne błędy. Należą do nich wewnętrzne i zewnętrzne błędy sprzętowe, takie jak błędy pamięci, uszkodzenie lub niewłaściwe funkcjonowanie jakiegoś urządzenia, a także różne błędy programowe, takie jak przepełnienie arytmetyczne, próba dostępu do wzbronionych lokacji w pamięci oraz niezdolność systemu operacyjnego do spełnienia wymagań aplikacji. W każdym przypadku system operacyjny musi zareagować tak, aby usunąć błąd przy możliwie

najmniejszym wpływie na czynne aplikacje. Reakcja może sięgać od zakończenia programu powodującego błąd poprzez ponowną próbę zrealizowania operacji aż do zwykłego poinformowania aplikacji o błędzie.

- **Ewidencjonowanie.** Dobry system operacyjny gromadzi dane statystyczne dotyczące wykorzystania zasobów i monitoruje parametry wydajnościowe, takie jak czas reakcji. W każdym systemie informacje te są użyteczne, co wynika z potrzeby przyszłych udoskonaleń i dostrajania systemu w celu zwiększenia jego wydajności. W systemie o wielu użytkownikach informacje te mogą służyć celom billingu.

System operacyjny jako program zarządzający zasobami

Komputer jest zbiorem zasobów służących do przenoszenia, przechowywania i przetwarzania danych oraz do sterowania tymi funkcjami. System operacyjny jest odpowiedzialny za zarządzanie tymi zasobami.

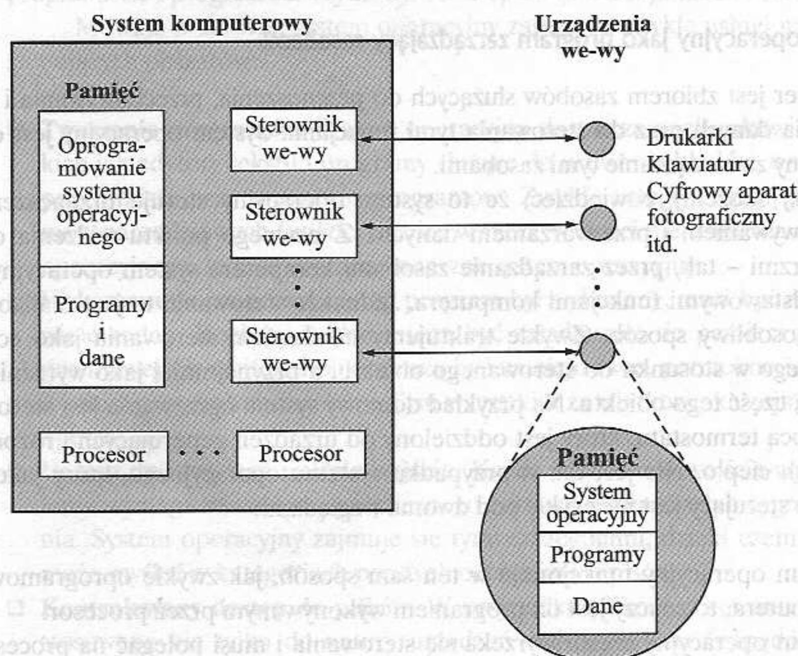
Czy możemy powiedzieć, że to system operacyjny steruje przenoszeniem, przechowywaniem i przetwarzaniem danych? Z pewnego punktu widzenia odpowiedź brzmi – tak, przez zarządzanie zasobami komputera system operacyjny steruje podstawowymi funkcjami komputera. Jednakże sterowanie to jest realizowane w dość osobliwy sposób. Zwykle traktujemy mechanizm sterowania jako coś zewnętrznego w stosunku do sterowanego obiektu, a przynajmniej jako wyraźnie wydzieloną część tego obiektu. Na przykład domowy system ogrzewania jest sterowany za pomocą termostatu, który jest oddzielony od urządzeń generujących i rozprowadzających ciepło. Nie jest tak w przypadku systemu operacyjnego, który jako mechanizm sterujący jest niezwykły pod dwoma względami:

- System operacyjny funkcjonuje w ten sam sposób, jak zwykłe oprogramowanie komputera; to znaczy jest on programem wykonywanym przez procesor.
- System operacyjny często wyrzeka się sterowania i musi polegać na procesorze, aby odzyskać sterowanie.

System operacyjny nie jest w rzeczywistości niczym więcej, jak tylko programem komputerowym. Podobnie jak inne programy komputerowe dostarcza on rozkazy do procesora. Jedyną różnicą jest intencja programu. System operacyjny kieruje procesorem w zakresie używania pozostałych zasobów systemu oraz synchronizowania wykonywania przez procesor innych programów. Żeby jednak procesor mógł wykonywać te czynności, musi zaprzestać realizowania programu systemu operacyjnego i zacząć wykonywać inne programy. System operacyjny wyrzeka się więc sterowania procesorem, umożliwiając mu wykonanie użytecznej pracy, po czym wznawia sterowanie z wyprzedzeniem wystarczającym do przygotowania następnej pracy. Wykorzystywany tu mechanizm zostanie wyjaśniony w dalszej części rozdziału.

Na rysunku 8.2 są pokazane główne zasoby, którymi zarządza system operacyjny. Część systemu operacyjnego znajduje się w pamięci głównej. Należy do niej **jądro** (*kernel* lub *nucleus*), które zawiera najczęściej używane funkcje systemu operacyjnego, oraz – w określonym momencie – inne części systemu operacyjnego uży-

wane na bieżąco. Pozostała część pamięci głównej zawiera inne programy i dane. Przydzielanie tych zasobów (pamięci głównej) jest sterowane wspólnie przez system operacyjny i sprzętowy mechanizm zarządzania pamięcią. System operacyjny decyduje, kiedy może być użyte urządzenie wejścia-wyjścia oraz steruje dostępem do plików i ich wykorzystaniem. Sam procesor jest pewnego rodzaju zasobem; system operacyjny musi określać, ile czasu procesora ma być poświęcone na realizację określonego programu użytkownika. W systemach wieloprocesorowych decyzja ta musi obejmować wszystkie procesory.



Rysunek 8.2. System operacyjny jako zarządca zasobów

Rodzaje systemów operacyjnych

Do różnicowania systemów operacyjnych służą pewne kluczowe własności. Własności te mogą być rozpatrywane w dwóch niezależnych wymiarach. W pierwszym wymiarze precyzuje się, czy system jest wsadowy, czy konwersacyjny. W przypadku systemu *konwersacyjnego* (*interactive*) ma miejsce konwersacja między użytkownikiem (programistą) a komputerem, zwykle za pośrednictwem terminalu klawiatura/monitor, mająca na celu zgłoszenie zapotrzebowania na wykonanie określonej pracy lub przeprowadzenie transakcji. Ponadto użytkownik może, zależnie od natury zastosowania, komunikować się z komputerem podczas wykonywania zadania. System *wsadowy* (*batch*) jest przeciwieństwem systemu konwersacyjnego. Programy wielu użytkowników są łączone (grupowane); powstaje wsad, który jest uruchamiany przez operatora komputera. Po zakończeniu wykonania programów wyniki są

drukowane i przekazywane użytkownikom. Czysto wsadowe systemy są dzisiaj rzadkością. Jednak zwięzłe przeanalizowanie systemów wsadowych będzie użyteczne dla przedstawienia współczesnych systemów operacyjnych.

W drugim, niezależnym wymiarze precyzuje się, czy system stosuje *wieloprogramowanie*, czy nie. Za pomocą wieloprogramowania czynione jest staranie o maksymalne możliwe obciążenie procesora pracą, poprzez jednoczesne wykonywanie więcej niż jednego programu. Do pamięci ładuje się kilka programów, a procesor „przeskakuje" szybko między nimi. Alternatywą jest system *jednoprogramowy*, który w określonym czasie wykonuje tylko jeden program.

Wczesne systemy

W przypadku najwcześniejszych komputerów, od końca lat czterdziestych do połowy pięćdziesiątych, programista współpracował bezpośrednio ze sprzętem komputerowym. Procesory były kierowane z konsoli, zawierającej lampki wskaźnikowe, przełączniki dwustabilne, pewną postać urządzenia wejściowego i drukarkę. Programy w kodzie maszynowym były ładowane przez urządzenie wejściowe (np. czytnik kart). Jeśli błąd spowodował zatrzymanie programu, było to sygnalizowane przez lampki.

Programista mógł wówczas analizować zawartość rejestrów i pamięci głównej, aby odnaleźć przyczynę błędu. Jeśli program był realizowany aż do normalnego zakończenia, wyniki ukazywały się w drukarce.

W tych wczesnych systemach występowały dwa główne problemy:

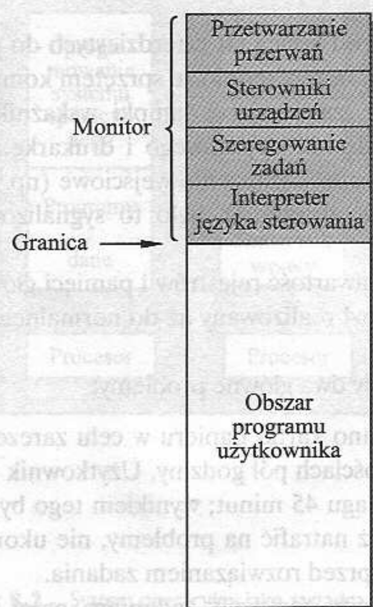
- **Planowanie.** W większości systemów używano kartki papieru w celu zarezerwowania czasu maszyny, zwykle w wielokrotnościach pół godziny. Użytkownik mógł wystąpić o godzinę, a zakończyć pracę w ciągu 45 minut; wynikiem tego był tracony czas komputera. Użytkownik mógł też natrafić na problemy, nie ukończyć pracy w przydzielonym czasie i przerwać ją przed rozwiązaniem zadania.
- **Czas na przygotowania.** Pojedynczy program, nazywany **zadaniem**, mógł obejmować ładowanie do pamięci kompilatora oraz programu w języku wysokiego poziomu (programu źródłowego), zapisanie programu skompilowanego (programu wynikowego), a następnie ładowanie oraz łączenie ze sobą programu wynikowego i powszechnie używanych funkcji. Każdy z tych kroków mógł zawierać zakładanie i wyjmowanie taśm oraz przygotowywanie paczek kart. Jeśli wystąpił błąd, nieszczęsny użytkownik musiał zwykle zaczynać przygotowania od początku. W rezultacie tracono dużo czasu na przygotowanie programu do pracy.

Taki tryb pracy mógłby być określony jako przetwarzanie szeregowe, co odzwierciedlałoby fakt, że dostęp użytkowników do komputera następuje szeregowo. Z czasem powstały różne narzędzia programowe podnoszące efektywność przetwarzania szeregowego. Należą do nich biblioteki powszechnie używanych funkcji, programy łączące, wprowadzające, usuwające błędy i procedury sterowania wejściem-wyjściem. Stały się one powszechnie dostępne dla wszystkich użytkowników.

Proste systemy wsadowe

Wczesne procesory były bardzo kosztowne, dlatego tak ważne było maksymalne ich wykorzystanie. Czas tracony z powodu planowania i przygotowań był nie do zaakceptowania.

W celu poprawienia wykorzystania opracowano proste wsadowe systemy operacyjne. W takim systemie, zwanym również *monitorem*, użytkownik nie miał już możliwości bezpośredniego dostępu do maszyny. Zamiast tego przekazywał zadanie w postaci kart lub taśmy operatorowi komputera, który porządkował sekwencyjnie zadania, tworząc wsad, po czym umieszczał wsad w urządzeniu wejściowym, do wykorzystania przez monitor (program zarządzający).



Rysunek 8.3. Rozkład pamięci w przypadku rezydentnego programu zarządzającego (monitora)

Aby zrozumieć funkcjonowanie tego schematu, rozpatrzymy jego działanie z dwóch punktów widzenia: programu zarządzającego i procesora. Z punktu widzenia programu zarządzającego to on właśnie steruje sekwencją zdarzeń. Żeby tak mogło być, program ten zawsze znajduje się w pamięci głównej i jest gotowy do wykonywania (rys. 8.3). Ta część programu jest nazywana **rezydentnym programem zarządzającym**. Pozostała część monitora składa się z programów narzędziowych i powszechnie używanych funkcji ładowanych jako podprogramy standardowe do programu użytkowego na początku każdego zadania, które tego wymaga. Program zarządzający wczytuje po jednym zadania z urządzenia wejściowego (zwykle czytnika kart lub napędu taśmowego). Po wczytaniu bieżące zadanie jest lokowane w obszarze programu użytkowego, a sterowanie jest przekazywane temu właśnie zadaniu. Gdy zadanie jest zakończone, następuje przerwanie (wewnętrzne w stosunku

do procesora), które powoduje zwrócenie sterowania programowi zarządzającemu. Ten z kolei natychmiast wczytuje następne zadanie. Wyniki każdego zadania są drukowane i dostarczane użytkownikowi.

Rozważmy teraz tę sekwencję z punktu widzenia procesora. W pewnym momencie procesor wykonuje rozkazy pochodzące z tej części pamięci głównej, w której jest zawarty program zarządzający. Rozkazy te sprawiają, że do innej części pamięci jest wczytywane następne zadanie. Po wczytaniu zadania procesor napotyka w programie zarządzającym rozkaz rozgałęzienia, który nakazuje mu kontynuowanie pracy określonej w innej komórce pamięci (początek programu użytkownika). Procesor następnie wykonuje rozkazy programu użytkownika, aż do zakończenia lub natrafienia na błąd. Dowolne z tych zdarzeń skłoni procesor do pobrania następnego rozkazu z programu zarządzającego. Tak więc zdanie „sterowanie zostało przekazane zadaniu" oznacza po prostu, że procesor pobiera teraz i wykonuje rozkazy pochodzące z programu użytkownika. Natomiast zdanie „sterowanie jest zwrócone programowi zarządzającemu" oznacza, że procesor pobiera i wykonuje rozkazy z programu zarządzającego.

Widoczne jest, że program zarządzający rozwiązuje problem planowania (szeregowania). Zadania stanowiące wsad są ustawiane w kolejce i są realizowane tak szybko, jak to jest możliwe, bez czasu jałowego związanego z interwencjami.

Co się natomiast dzieje z problemem przygotowywania? Również i to rozwiązuje program zarządzający. Każdemu zadaniu towarzyszą rozkazy sformułowane w **języku sterowania zadaniami (JCL)**. Jest to specjalny rodzaj języka programowania używany do dostarczania instrukcji programowi zarządzającemu. W prostym przykładzie użytkownik dostarcza program napisany w Fortranie oraz dane przeznaczone do wykorzystania przez program. Każda instrukcja Fortranu i każda pozycja danych znajdują się na oddzielnych kartach dziurkowanych lub w oddzielnym rekordzie na taśmie. Obok wierszy Fortranu i danych, zadanie zawiera rozkazy sterowania zadaniami, oznaczone na początku znakiem $. Ogólny format zadania może wyglądać następująco:

```
$JOB
$FTN
•
•  } Instrukcje Fortranu
•
$LOAD
$RUN
•
•  } Dane
•
$END
```

W celu wykonania tego zadania monitor wczytuje wiersz $FTN i ładuje odpowiedni kompilator ze swojej pamięci masowej (zwykle taśmowej). Kompilator tłumaczy program użytkowy na kod wynikowy, który zostaje zapisany w pamięci głównej lub masowej. Jeśli zostaje zapisany w pamięci głównej, następnym wymaganym rozkazem jest $LOAD. Rozkaz ten jest wczytywany przez program zarządzający, który ponow-

nie przejmuje sterowanie po operacji kompilowania. Program zarządzający uruchamia jednostkę ładowania, która ładuje program wynikowy do pamięci w miejsce kompilatora i przekazuje mu sterowanie. W ten sposób duży segment pamięci głównej może być użytkowany wspólnie przez różne podsystemy, chociaż w danej chwili tylko jeden podsystem może rezydować w pamięci i wykonywać program.

Widzimy, że program zarządzający (lub wsadowy system operacyjny) jest po prostu programem komputerowym. Umożliwia on procesorowi pobieranie rozkazów z różnych części pamięci głównej w celu albo przejmowania sterowania, albo rezygnowania z niego. Wymagane są również pewne inne cechy sprzętowe:

- **Ochrona pamięci.** Wykonywanie programu użytkownika nie może powodować zmian w obszarze pamięci zajętym przez program zarządzający. Jeśli występuje takie usiłowanie, to procesor wykrywa błąd i przekazuje sterowanie programowi zarządzającemu. Program ten porzuca zadanie, drukuje komunikat o błędzie i ładuje następne zadanie.
- **Czasomierz.** Czasomierz jest używany dla zapobiegania monopolizacji systemu przez pojedyncze zadanie. Jest on ustawiany na początku każdego zadania. Po dojściu do określonego wskazania, następuje przerwanie i sterowanie wraca do programu zarządzającego.
- **Rozkazy uprzywilejowane.** Pewne rozkazy są oznaczone jako uprzywilejowane i mogą być wykonywane tylko przez program zarządzający. Należą do nich rozkazy wejścia-wyjścia, a więc program zarządzający zachowuje sterowanie wszystkimi urządzeniami wejścia-wyjścia. Zapobiega to na przykład temu, żeby program użytkownika przypadkowo przyjął rozkazy sterowania pochodzące z następnego zadania. Jeśli z programu użytkownika wynika potrzeba operacji wejścia-wyjścia, to musi on zwrócić się do programu zarządzającego, aby ten wykonał w jego imieniu tę operację. Jeśli procesor napotyka rozkaz uprzywilejowany podczas wykonywania programu użytkownika, to traktuje go jako błąd i przekazuje sterowanie programowi zarządzającemu.
- **Przerwania.** Wczesne modele komputerów nie dysponowały tą zdolnością. Właściwość ta daje systemowi operacyjnemu większą elastyczność, jeśli chodzi o rezygnowanie ze sterowania i przejmowanie go od programów użytkowych.

Czas maszynowy jest dzielony między wykonywanie programów użytkownika a wykonywanie programu zarządzającego. Towarzyszą temu dwie straty: część pamięci głównej jest oddana programowi zarządzającemu, a pewna część czasu maszynowego jest zużywana przez ten program. Nawet z tymi stratami stosowanie prostego systemu wsadowego poprawia wykorzystanie komputera.

Wieloprogramowe systemy wsadowe

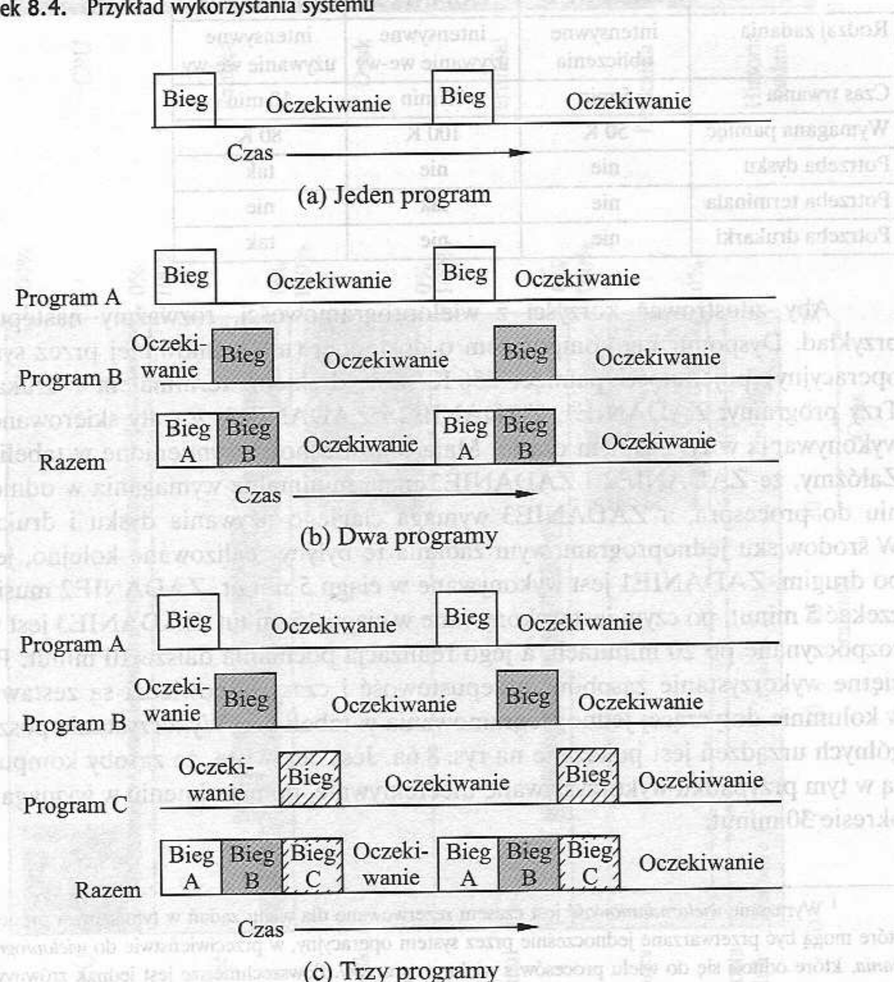
Nawet przy automatycznym szeregowaniu zadań realizowanym przez prosty wsadowy system operacyjny procesor często pozostaje niewykorzystany. Problem polega na tym, że urządzenia wejścia-wyjścia są powolne w porównaniu z procesorem. Na rysunku 8.4 są przedstawione reprezentatywne obliczenia. Dotyczą one programu

który przetwarza pliki rekordów i wykonuje przeciętnie 100 rozkazów maszynowych na 1 rekord. W tym przykładzie komputer spędza ponad 96% czasu, czekając na zakończenie transferu danych przez urządzenia wejścia-wyjścia! Na rysunku 8.5a zilustrowano tę sytuację. Procesor przez pewien czas wykonuje rozkazy, aż napotka rozkaz wejścia-wyjścia. Musi następnie czekać na zakończenie wykonywania rozkazu wejścia-wyjścia, zanim podejmie dalszą pracę.

Odczytanie jednego rekordu z pliku	0,0015 s
Wykonanie 100 rozkazów	0,0001 s
Zapisanie jednego rekordu w pliku	0,0015 s
RAZEM	0,0031 s

$$\text{Procent wykorzystania procesora} = \frac{0{,}0001}{0{,}0031} = 0{,}032 = 3{,}2\%$$

Rysunek 8.4. Przykład wykorzystania systemu



Rysunek 8.5. Przykład wieloprogramowania

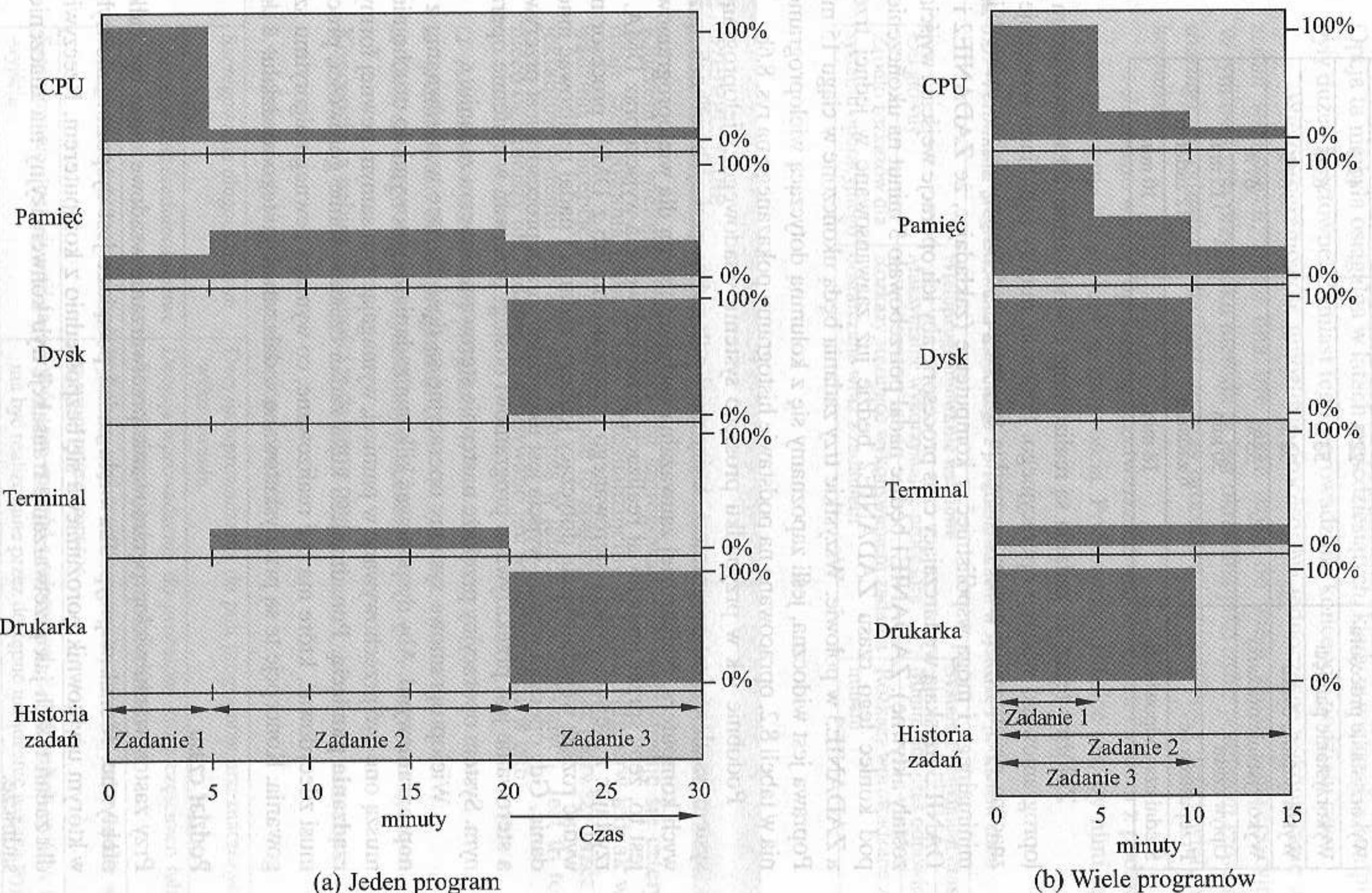
Ta nieefektywność nie jest konieczna. Wiemy już, że pojemność pamięci musi być wystarczająca do pomieszczenia rezydentnego programu zarządzającego i jednego programu użytkownika. Załóżmy, że pojemność byłaby wystarczająca dla systemu operacyjnego i dwóch programów użytkowych. W takim przypadku, gdy jedno z zadań musi czekać na wejście lub wyjście, procesor może się przełączyć na inne zadanie, które prawdopodobnie nie czeka na wejście-wyjście (rys. 8.5b). Ponadto możemy rozszerzyć pamięć, umożliwiając zmieszczenie trzech, czterech lub większej liczby programów, po czym przełączać się między nimi (rys. 8.5c). Proces ten jest znany jako **wieloprogramowanie** (*multiprogramming*) lub **wielozadaniowość** (*multitasking*)[1]. Jest on głównym zagadnieniem nowoczesnych systemów operacyjnych.

Tabela 8.1. Właściwości wykonywania przykładowego programu

	ZADANIE 1	ZADANIE 2	ZADANIE 3
Rodzaj zadania	intensywne obliczenia	intensywne używanie we-wy	intensywne używanie we-wy
Czas trwania	5 min	15 min	10 min
Wymagana pamięć	50 K	100 K	80 K
Potrzeba dysku	nie	nie	tak
Potrzeba terminala	nie	tak	nie
Potrzeba drukarki	nie	nie	tak

Aby zilustrować korzyści z wieloprogramowości, rozważmy następujący przykład. Dysponujemy komputerem o dostępnej (nie zajmowanej przez system operacyjny) pojemności pamięci 256 K słów, dyskiem, terminalem i drukarką. Trzy programy: ZADANIE1, ZADANIE2 i ZADANIE3, zostały skierowane do wykonywania w tym samym czasie. Mają one własności wymienione w tabeli 8.1. Załóżmy, że ZADANIE2 i ZADANIE3 mają minimalne wymagania w odniesieniu do procesora, a ZADANIE3 wymaga ciągłego używania dysku i drukarki. W środowisku jednoprogramowym zadania te byłyby realizowane kolejno, jedno po drugim. ZADANIE1 jest wykonywane w ciągu 5 minut. ZADANIE2 musi odczekać 5 minut, po czym jest wykonywane w ciągu 15 minut. ZADANIE3 jest więc rozpoczynane po 20 minutach, a jego realizacja pochłania dalsze 10 minut. Przeciętne wykorzystanie zasobów, przepustowość i czasy odpowiedzi są zestawione w kolumnie dotyczącej jednoprogramowania w tabeli 8.2. Wykorzystanie poszczególnych urządzeń jest pokazane na rys. 8.6a. Jest oczywiste, że zasoby komputera są w tym przypadku wykorzystywane nieefektywnie, po uśrednieniu w wymaganym okresie 30 minut.

[1] Wyrażenie *wielozadaniowość* jest czasem rezerwowane dla wielu zadań w tym samym programie, które mogą być przetwarzane jednocześnie przez system operacyjny, w przeciwieństwie do *wieloprogramowania,* które odnosi się do wielu procesów z wielu programów. Powszechniejsze jest jednak zrównywanie znaczeń *wielozadaniowości* i *wieloprogramowania*, co jest czynione w większości słowników normalizacyjnych (np. IEEE Std 100-1992, *The New IEEE Standard Dictionary of Electrical and Electronics Terms*).



Rysunek 8.6. Histogram wykorzystania

Tabela 8.2. Wpływ wieloprogramowania na wykorzystanie zasobów

	Przetwarzanie jednoprogramowe	Wieloprogramowanie
Wykorzystanie procesora	22%	43%
Wykorzystanie pamięci	33%	63%
Wykorzystanie dysku	33%	67%
Wykorzystanie drukarki	33%	67%
Upływ czasu	30 min	15 min
Przepustowość	6 zadań/h	12 zadań/h
Średni czas odpowiedzi	18 min	10 min

Załóżmy teraz, że zadania są realizowane równocześnie, z zastosowaniem wieloprogramowego systemu operacyjnego. Ponieważ między zadaniami występuje nieznaczna rywalizacja o zasoby, wszystkie trzy mogą być realizowane w czasie bliskim minimalnego i mogą współistnieć w komputerze (zakładając, że ZADANIE2 i ZADANIE3 uzyskają wystarczający czas procesora, aby ich operacje wejścia i wyjścia pozostały aktywne). ZADANIE1 będzie nadal potrzebowało 5 minut na ukończenie, ale pod koniec tego czasu ZADANIE2 będzie już zaawansowane w jednej trzeciej, a ZADANIE3 w połowie. Wszystkie trzy zadania będą ukończone w ciągu 15 minut. Poprawa jest widoczna, jeśli zapoznamy się z kolumną dotyczącą wieloprogramowania w tabeli 8.2, opracowaną na podstawie histogramu pokazanego na rys. 8.6b.

Podobnie jak w przypadku prostego systemu wsadowego, wieloprogramowy system wsadowy jest programem, który musi polegać na pewnych cechach sprzętowych komputera. Najbardziej zauważalną cechą użyteczną dla wieloprogramowości jest to, żeby sprzęt umożliwiał realizację przerwań wejścia-wyjścia oraz DMA. Korzystając z wejścia-wyjścia sterowanego przerwaniami lub z DMA, procesor może wydać rozkaz wejścia-wyjścia dotyczący jednego zadania i nadal realizować inne zadanie. Gdy operacja wejścia-wyjścia jest ukończona, praca procesora jest przerywana, a sterowanie jest przekazywane programowi obsługi przerwań w systemie operacyjnym. System operacyjny przekaże następnie sterowanie innemu zadaniu.

Wieloprogramowe systemy operacyjne są dość złożone w porównaniu z **jednoprogramowymi**. Aby dysponować kilkoma zadaniami gotowymi do uruchomienia, muszą one być przechowywane w pamięci, wymagając tym samym pewnej formy **zarządzania pamięcią**. Ponadto, jeśli kilka zadań czeka w stanie gotowości, procesor musi zdecydować, które ma być realizowane, co wymaga pewnego algorytmu szeregowania. Koncepcje te są przedyskutowane w dalszym ciągu tego rozdziału.

Podział czasu

Przy zastosowaniu wieloprogramowania przetwarzanie wsadowe może być całkiem efektywne. Jednak w przypadku wielu zadań pożądane jest dysponowanie trybem, w którym użytkownik porozumiewa się bezpośrednio z komputerem. Rzeczywiście, dla zadań takich jak przetwarzanie transakcji tryb konwersacyjny ma znaczenie zasadnicze.

W dzisiejszych czasach wymaganie przetwarzania konwersacyjnego może być – i często jest – zaspokajane przez użycie mikrokomputerów wyspecjalizowanych. Opcja ta nie była osiągalna w latach sześćdziesiątych, kiedy w większości komputery były duże i kosztowne. Zamiast tego wprowadzono koncepcję podziału czasu.

Wieloprogramowość umożliwia procesorowi jednoczesne wykonywanie wielu zadań wsadowych; może ona być także użyta do wielu zadań konwersacyjnych. W tym ostatnim przypadku metoda ta jest określana jako *tryb z podziałem czasu* (*time sharing*), co odzwierciedla fakt, że czas procesora jest dzielony między wielu użytkowników.

Zarówno wieloprogramowe przetwarzanie wsadowe, jak i system z podziałem czasu posługują się wieloprogramowością. Podstawowe różnice między nimi są wymienione w tabeli 8.3.

Tabela 8.3. Porównanie programowania wsadowego z programowaniem w systemie z podziałem czasu

	Wieloprogramowanie wsadowe	System z podziałem czasu
Cel główny	Maksymalne wykorzystanie procesora	Minimalny czas odpowiedzi
Źródło rozkazów dla systemu operacyjnego	Rozkazy sterujące wykonywaniem zadania dostarczane wraz z zadaniem	Rozkazy wprowadzane z terminalu

8.2. Szeregowanie

Kluczem do wieloprogramowania jest szeregowanie. Zwykle stosuje się cztery rodzaje szeregowania (tabela 8.4). Przeanalizujemy je teraz. Najpierw jednak wprowadzimy koncepcję *procesu*. Wyrażenie to zostało użyte po raz pierwszy przez projektantów systemu operacyjnego Multics w latach sześćdziesiątych. Jest to termin nieco ogólniejszy niż zadanie. Przypisywano mu wiele definicji, w tym:

- Aktualnie wykonywany program.
- „Animowany duch" programu.
- Coś, do czego został przydzielony procesor.

Koncepcja ta nabierze jasności w dalszym ciągu.

Tabela 8.4. Szeregowanie w przypadku wieloprogramowania

Szeregowanie długookresowe	Decyzja wprowadzenia do puli programów przeznaczonych do wykonywania
Szeregowanie średniookresowe	Decyzja wprowadzenia do procesów, które częściowo lub całkowicie znajdują się w pamięci głównej
Szeregowanie krótkookresowe	Wybór tego spośród dostępnych procesów, który będzie wykonywany przez procesor
Szeregowanie operacji wejścia-wyjścia	Wybór tego spośród zawieszonych żądań wejścia-wyjścia, które ma być realizowane przez dostępne urządzenie wejścia-wyjścia

Szeregowanie długookresowe

Długookresowy program szeregujący wysokiego poziomu określa, które programy są dopuszczone do przetwarzania przez system. Program ten steruje więc stopniem wieloprogramowości (liczbą procesów w pamięci). Po dopuszczeniu zadanie lub program staje się procesem i jest dodawane do kolejki zarządzanej przez krótkookresowy program szeregujący. W niektórych systemach nowo utworzony proces rozpoczyna się w stanie usunięcia z pamięci; wówczas jest on wprowadzany do kolejki średnioterminowego programu szeregującego.

W systemie wsadowym lub w przypadku części wsadowej systemu operacyjnego ogólnego przeznaczenia nowe zadania są kierowane na dysk i trzymane w kolejce. Program szeregujący wysokiego poziomu pobiera zadania z kolejki, gdy jest to możliwe. Muszą tu być podjęte dwie decyzje. Po pierwsze, program szeregujący musi zdecydować, czy może pobrać jeden, czy wiele dodatkowych procesów. Po drugie, program szeregujący musi zdecydować, które zadanie lub zadania mają być akceptowane i zamienione na procesy. Stosowane kryteria mogą obejmować priorytet, czas realizacji i wymagania co do wejścia-wyjścia.

W przypadku programów konwersacyjnych w systemie podziału czasu zapotrzebowanie na proces jest generowane przez działanie użytkownika, który chce się dołączyć do systemu. Użytkownicy systemu z podziałem czasu nie są po prostu ustawiani w kolejce i trzymani w niej, aż do momentu akceptacji ich przez system. System operacyjny będzie raczej akceptował wszystkich autoryzowanych użytkowników aż do nasycenia systemu. Wówczas żądanie dołączenia się do systemu spotka się z odpowiedzią, że system jest w pełni wykorzystany, a użytkownik powinien spróbować ponownie później.

Szeregowanie średniookresowe

Szeregowanie średniookresowe jest częścią funkcji wymiany opisanej w podrozdz. 8.3. Zwykle decyzja wczytania do pamięci wynika z potrzeby zarządzania stopniem wieloprogramowości. W systemie nie korzystającym z pamięci wirtualnej powodem może być również zarządzanie pamięcią. Zatem decyzja wczytania musi uwzględniać wymagania pamięciowe procesów usuniętych z pamięci.

Szeregowanie krótkookresowe

Program szeregujący wysokiego poziomu działa stosunkowo rzadko i podejmuje ogólne decyzje, czy przyjąć nowy proces i który proces ma być przyjęty. Krótkookresowy program szeregujący, znany również jako *dyspozytor* (*dispatcher*), działa często i podejmuje bardziej szczegółowe decyzje, które zadanie ma być realizowane jako następne.

Stany procesu

Aby zrozumieć działanie dyspozytora, musimy rozważyć koncepcję stanu procesu. Podczas czasu życia procesu, jego status zmienia się wielokrotnie. Status procesu

w dowolnym czasie jest nazywany *stanem*. Użyty został termin „stan", ponieważ oznacza on jednocześnie, że istnieje pewna informacja definiująca status w tym momencie. Definiuje się zwykle pięć stanów procesu (rys. 8.7):

- **Nowy.** Program jest przyjęty przez program szeregujący wysokiego poziomu, lecz nie jest gotowy do realizacji. System operacyjny inicjuje proces, przesuwając go w ten sposób do stanu gotowości.
- **Gotowy.** Proces jest gotowy do wykonywania i czeka na dostęp do procesora.
- **Bieżący.** Proces jest realizowany przez procesor.
- **Oczekujący.** Realizacja procesu jest zawieszona podczas oczekiwania na pewne zasoby systemu, np. wejście-wyjście.
- **Zatrzymany.** Proces został zakończony i zostanie zniszczony przez system operacyjny.



Rysunek 8.7. Model procesu pięciostanowego

Dla każdego procesu w systemie system operacyjny musi utrzymywać informację o stanie, wskazującą na status procesu oraz na inne cechy istotne do realizacji procesu. W tym celu każdy proces jest reprezentowany w systemie operacyjnym przez *blok kontrolny procesu* (rys. 8.8). W bloku kontrolnym procesu są zwykle zawarte:

- **Identyfikator.** Każdy aktualny proces ma unikatowy identyfikator.
- **Stan.** Aktualny stan procesu (nowy, gotowy itd.).
- **Priorytet.** Względny poziom priorytetu.
- **Licznik programu.** Adres następnego rozkazu w programie, który ma być wykonywany.
- **Znaczniki pamięci.** Początkowa i końcowa komórka zajmowane przez proces w pamięci.
- **Dane dotyczące kontekstu.** Są to dane obecne w rejestrach procesora podczas realizacji procesu; będą one przedyskutowane w części III. Teraz wystarczy stwierdzić, że dane te reprezentują „kontekst" procesu. Kontekst i licznik programu są zachowywane, gdy proces przestaje być w stanie gotowości. Są pobierane przez procesor, gdy wznawia on realizację procesu.

Identyfikator
Stan
Priorytet
Licznik rozkazów
Wskaźniki pamięci
Dane kontekstowe
Informacje o stanie we-wy
Informacja ewidencyjna
⋮

Rysunek 8.8. Blok kontrolny procesu

- **Informacja o stanie wejścia-wyjścia.** Obejmuje wyróżniające się żądania wejścia-wyjścia, urządzenia wejścia-wyjścia (np. napędy taśmowe) przypisane do procesu, listę plików przypisanych do procesu itd.
- **Informacja ewidencyjna.** Może obejmować wymagany czas procesora i czas zegara, ograniczenia czasowe, liczby ewidencyjne itd.

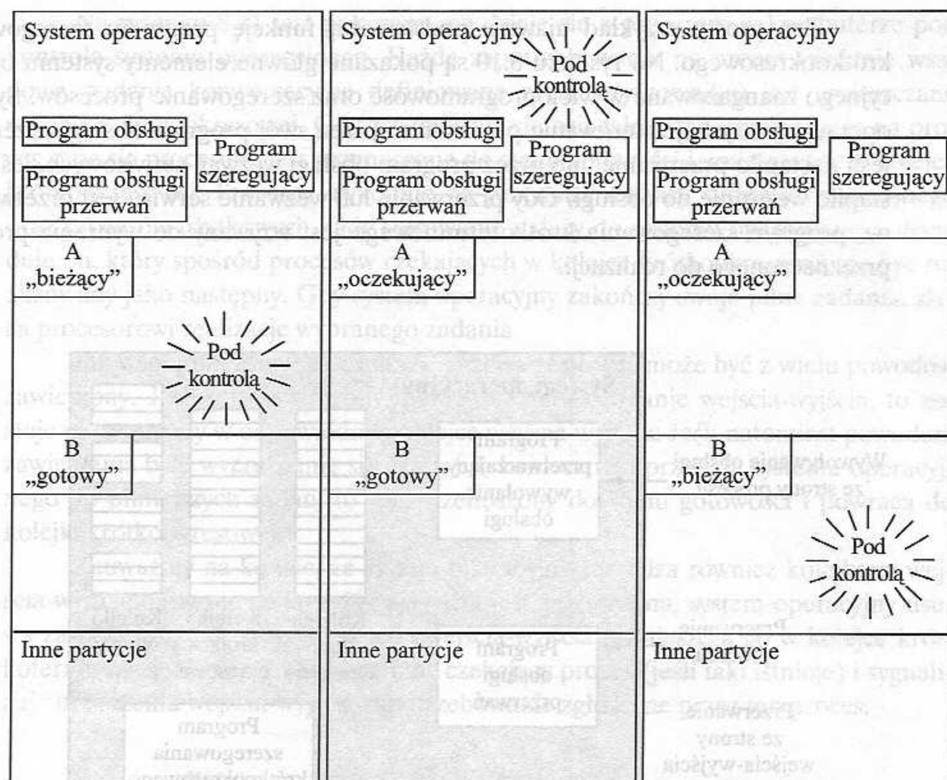
Gdy procesor akceptuje do wykonania nowe zadanie lub żądanie użytkownika, tworzy czysty blok kontrolny procesu i umieszcza związany z nim proces w stanie „nowy". Po wypełnieniu przez system bloku kontrolnego procesu, proces jest przenoszony do stanu „gotowy".

Metody szeregowania

Aby zrozumieć, jak system operacyjny zarządza szeregowaniem różnych zadań w pamięci, rozpocznijmy od rozważenia prostego przykładu z rys. 8.9. Na rysunku widać, jak w określonym momencie jest partycjonowana pamięć. Oczywiście, jądro systemu operacyjnego zawsze jest rezydentne. Ponadto występuje pewna liczba aktywnych procesów, w tym A i B, z których każdy jest rozmieszczony w części pamięci.

Rozpoczniemy w momencie, gdy jest realizowany proces A. Procesor wykonuje rozkazy programu zawartego w części pamięci przypisanej do A. W pewnym późniejszym momencie procesor zaprzestaje wykonywania rozkazów procesu A i rozpoczyna wykonywanie rozkazów przechowywanych w obszarze systemu operacyjnego. Następuje to z jednego z trzech powodów:

1. Proces A przekazuje wywołanie obsługi (np. żądanie wejścia-wyjścia) systemowi operacyjnemu. Wykonywanie procesu A jest więc zawieszone do czasu spełnienia żądania przez system operacyjny.

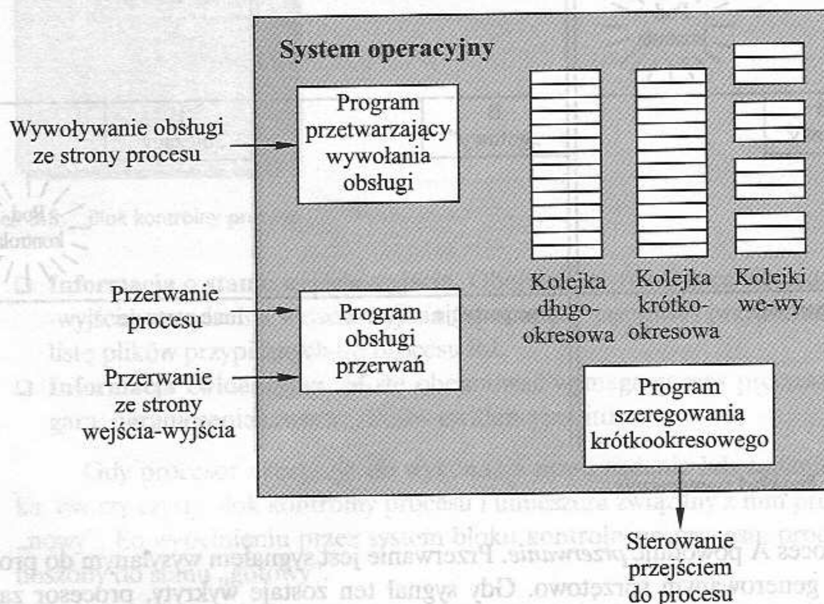


Rysunek 8.9. Przykład szeregowania

2. Proces A powoduje *przerwanie*. Przerwanie jest sygnałem wysyłanym do procesora generowanym sprzętowo. Gdy sygnał ten zostaje wykryty, procesor zaprzestaje wykonywania procesu A i przenosi się do programu obsługi przerwań w systemie operacyjnym. Różne zdarzenia związane z procesem A mogą spowodować przerwanie. Jedną z możliwości jest błąd, taki jak przystąpienie do wykonywania rozkazu uprzywilejowanego. Innym przykładem jest wyczerpanie się czasu; w celu zapobieżenia monopolizacji procesora przez jeden proces, każdy proces uzyskuje dostęp do procesora tylko na krótki czas.
3. Pewne zdarzenie nie związane z procesem A, lecz wymagające uwagi, powoduje przerwanie. Przykładem jest zakończenie operacji wejścia-wyjścia.

W każdym przypadku wynik jest następujący. Procesor zapisuje bieżące dane kontekstowe i licznik programu A w bloku kontrolnym procesu A, po czym przystępuje do działania w systemie operacyjnym. System operacyjny może wykonać pewną pracę, taką jak zainicjowanie operacji wejścia-wyjścia. Potem należący do systemu operacyjnego program szeregowania krótkookresowego decyduje, który proces powinien być realizowany jako następny. W tym przykładzie został wybrany proces B. System operacyjny poleca procesorowi odnowienie danych kontekstu B i przystąpienie do realizowania procesu B w punkcie, w którym był on pozostawiony.

Ten prosty przykład ujawnia podstawową funkcję programu szeregowania krótkookresowego. Na rysunku 8.10 są pokazane główne elementy systemu operacyjnego zaangażowane w wieloprogramowość oraz szeregowanie procesów. System operacyjny uzyskuje sterowanie procesorem przez swój program obsługi przerwań, jeśli nastąpiło przerwanie, lub przez program obsługi wezwań serwisowych, jeśli wystąpiło wezwanie do obsługi. Gdy przerwanie lub wezwanie serwisu jest przetwarzane, program szeregowania krótkoterminowego jest wzywany do wybrania procesu przeznaczonego do realizacji.



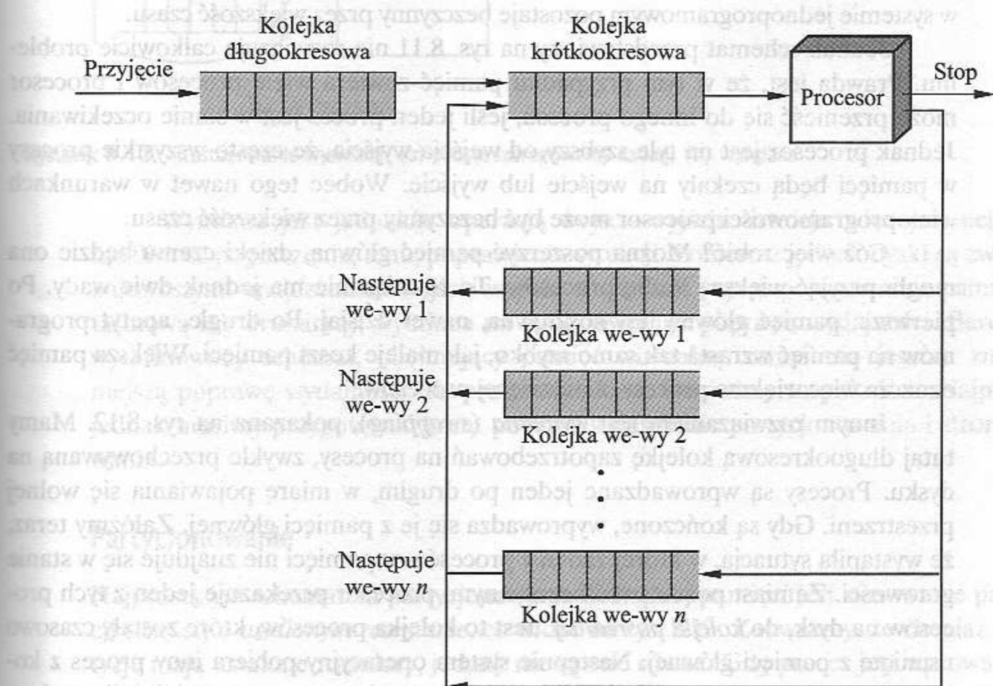
Rysunek 8.10. Główne składniki systemu operacyjnego w przypadku wieloprogramowania

W celu wykonania tego zadania system operacyjny utrzymuje pewną liczbę kolejek. Każda kolejka jest po prostu listą procesów czekających na pewne zasoby. *Kolejka długookresowa* jest listą zadań czekających na użycie systemu. Gdy warunki pozwolą, program szeregujący wysokiego poziomu dokona alokacji pamięci i utworzy proces z jednej spośród czekających jednostek. *Kolejka krótkookresowa* składa się ze wszystkich procesów będących w stanie gotowości. Każdy z tych procesów mógłby korzystać z procesora jako następny. Do programu szeregowania krótkookresowego należy wybranie jednego z nich. Na ogół jest to robione za pomocą algorytmu cyklicznego, co powoduje przydzielenie czasu kolejno każdemu procesowi. Mogą też być użyte poziomy priorytetu. Jest wreszcie *kolejka wejścia-wyjścia* związana z każdym urządzeniem wejścia-wyjścia. Wiele procesów może domagać się użycia tego samego urządzenia wejścia-wyjścia. Wszystkie procesy czekające na użycie określonego urządzenia są ustawiane w jego kolejce.

Na rysunku 8.11 jest pokazane, co dzieje się z procesem w komputerze pod kontrolą systemu operacyjnego. Każde zapotrzebowanie na proces (zadanie wsadowe, zadanie konwersacyjne definiowane przez użytkownika) jest umieszczane w kolejce długookresowej. Gdy zasoby stają się osiągalne, zapotrzebowanie na proces staje się procesem, jest przenoszone do stanu gotowości i umieszczane w kolejce krótkookresowej. Procesor wykonuje na przemian rozkazy systemu operacyjnego oraz procesów użytkowych. Gdy sterowanie należy do systemu operacyjnego, decyduje on, który spośród procesów czekających w kolejce krótkookresowej ma być realizowany jako następny. Gdy system operacyjny zakończy swoje pilne zadania, zleca procesorowi realizację wybranego zadania.

Jak wspomnieliśmy wcześniej, realizowany proces może być z wielu powodów zawieszony. Jeśli jest zawieszony, ponieważ zgłosił żądanie wejścia-wyjścia, to zostaje umieszczony w odpowiedniej kolejce wejścia-wyjścia. Jeśli natomiast powodem zawieszenia było wyczerpanie się czasu lub konieczność przejścia systemu operacyjnego do pilniejszych zadań, to jest przenoszony do stanu gotowości i powraca do kolejki krótkookresowej.

Zauważmy na koniec, że system operacyjny zarządza również kolejkami wejścia-wyjścia. Gdy operacja wejścia-wyjścia jest zakończona, system operacyjny usuwa zaspokojony już proces z kolejki wejścia-wyjścia i umieszcza go w kolejce krótkoterminowej. Następnie wybiera inny czekający proces (jeśli taki istnieje) i sygnalizuje urządzeniu wejścia-wyjścia zapotrzebowanie zgłoszone przez ten proces.



Rysunek 8.11. Diagram kolejek związanych z szeregowaniem pracy procesora

8.3. Zarządzanie pamięcią

W systemie jednoprogramowym pamięć główna jest podzielona na dwie części: jedną przeznaczoną dla systemu operacyjnego (rezydentnego programu zarządzającego) i drugą dla programu aktualnie realizowanego. W systemie wieloprogramowym część pamięci przypisana użytkownikowi musi być dalej podzielona w celu przyjęcia wielu procesów. Zadanie tego dalszego podziału jest realizowane dynamicznie przez system operacyjny i nosi nazwę *zarządzania pamięcią*.

Sprawne zarządzanie pamięcią jest niezwykle ważne w systemie wieloprogramowym. Jeśli w pamięci znajduje się niewiele procesów, to znaczną część czasu wszystkich procesów zabierze oczekiwanie na wejście-wyjście i procesor będzie „bezrobotny". Wobec tego pamięć musi być sprawnie zarządzana w celu upakowania w pamięci możliwie dużej liczby procesów.

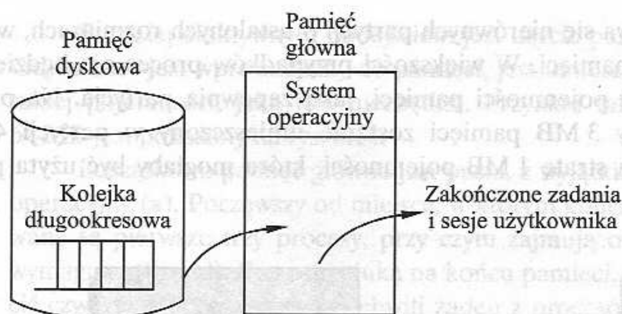
Wymiana

Wróćmy do rysunku 8.11. Przedyskutowaliśmy trzy rodzaje kolejek: długookresową kolejkę zapotrzebowań na nowe procesy, kolejkę krótkookresową procesów gotowych do użycia procesora i różne kolejki wejścia-wyjścia procesów, które nie są gotowe do użycia procesora. Zauważmy, że przyczyną tej rozbudowanej struktury jest to, że operacje wejścia-wyjścia są o wiele wolniejsze niż obliczenia i dlatego właśnie procesor w systemie jednoprogramowym pozostaje bezczynny przez większość czasu.

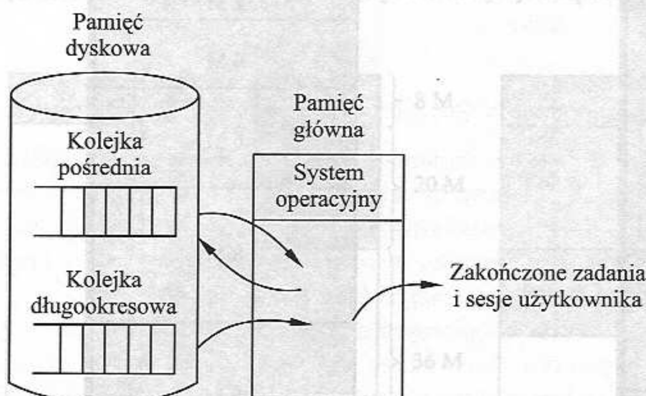
Jednak schemat przedstawiony na rys. 8.11 nie rozwiązuje całkowicie problemu. Prawdą jest, że w tym przypadku pamięć zawiera wiele procesów i procesor może przenieść się do innego procesu, jeśli jeden proces jest w stanie oczekiwania. Jednak procesor jest na tyle szybszy od wejścia-wyjścia, że często wszystkie procesy w pamięci będą czekały na wejście lub wyjście. Wobec tego nawet w warunkach wieloprogramowości procesor może być bezczynny przez większość czasu.

Cóż więc robić? Można poszerzyć pamięć główną, dzięki czemu będzie ona mogła przyjąć większą liczbę procesów. To rozwiązanie ma jednak dwie wady. Po pierwsze, pamięć główna jest kosztowna, nawet dzisiaj. Po drugie, apetyt programów na pamięć wzrasta tak samo szybko, jak maleje koszt pamięci. Większa pamięć oznacza więc większe procesy, a nie więcej procesów.

Innym rozwiązaniem jest *wymiana* (*swapping*), pokazana na rys. 8.12. Mamy tutaj długookresową kolejkę zapotrzebowań na procesy, zwykle przechowywaną na dysku. Procesy są wprowadzane jeden po drugim, w miarę pojawiania się wolnej przestrzeni. Gdy są kończone, wyprowadza się je z pamięci głównej. Załóżmy teraz, że wystąpiła sytuacja, w której żaden z procesów w pamięci nie znajduje się w stanie gotowości. Zamiast pozostawać bezczynnym, procesor przekazuje jeden z tych procesów na dysk, do *kolejki pośredniej*. Jest to kolejka procesów, które zostały czasowo usunięte z pamięci głównej. Następnie system operacyjny pobiera inny proces z kolejki pośredniej lub honoruje nowe zapotrzebowanie na proces z kolejki długookresowej, po czym realizowany jest nowo przybyły proces.



(a) Proste szeregowanie zadania



(b) Wymiana

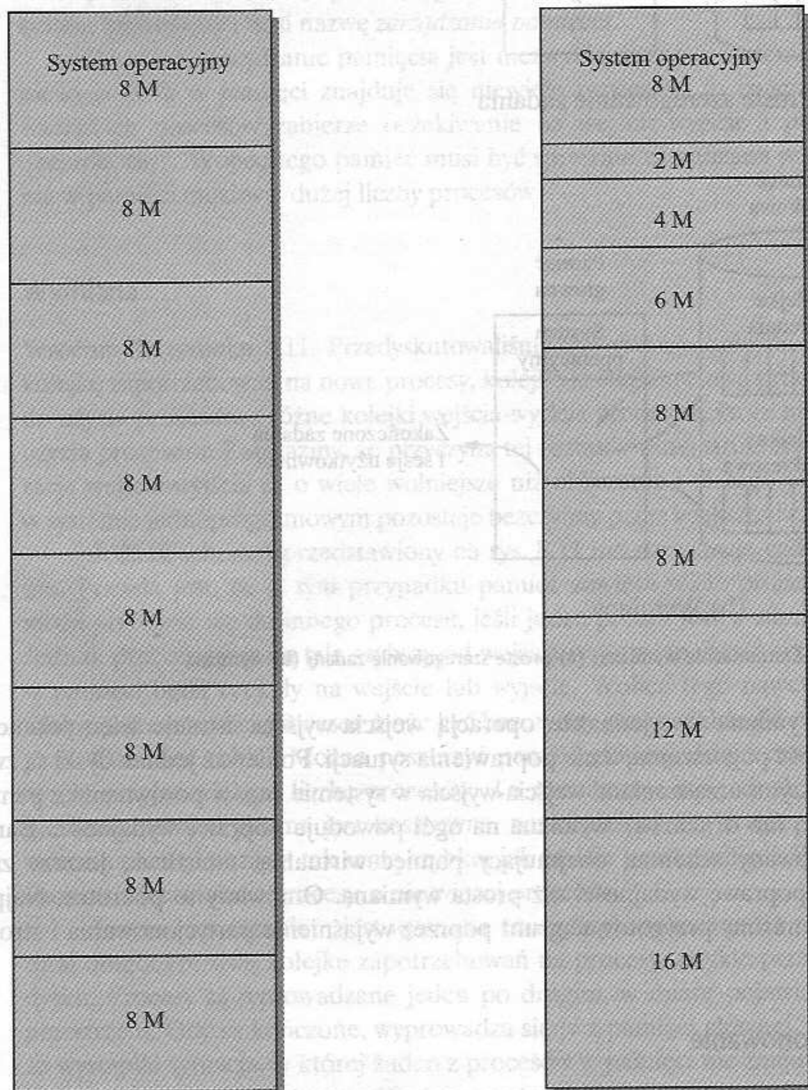
Rysunek 8.12. Zastosowanie wymiany: (a) proste szeregowanie zadań; (b) wymiana

Wymiana jest jednakże operacją wejścia-wyjścia, istnieje więc potencjalna możliwość pogorszenia, a nie poprawienia sytuacji. Ponieważ jednak dyski są zwykle najszybszymi urządzeniami wejścia-wyjścia w systemie (np. w porównaniu z pamięcią taśmową lub drukarką), wymiana na ogół powoduje poprawę wydajności. Bardziej wyrafinowany schemat, obejmujący pamięć wirtualną, umożliwia jeszcze znaczniejszą poprawę wydajności niż prosta wymiana. Omówimy to pokrótce. Najpierw jednak musimy przygotować grunt poprzez wyjaśnienie partycjonowania i stronicowania.

Partycjonowanie

Najprostszym schematem partycjonowania dostępnej pamięci jest stosowanie partycji (części) o *ustalonym rozmiarze*, co widać na rys. 8.13. Zauważmy, że chociaż partycje mają ustalone rozmiary, jednak nie są one równe. Gdy proces jest wprowadzany do pamięci, jest umieszczany w najmniejszej dostępnej partycji, która może go pomieścić.

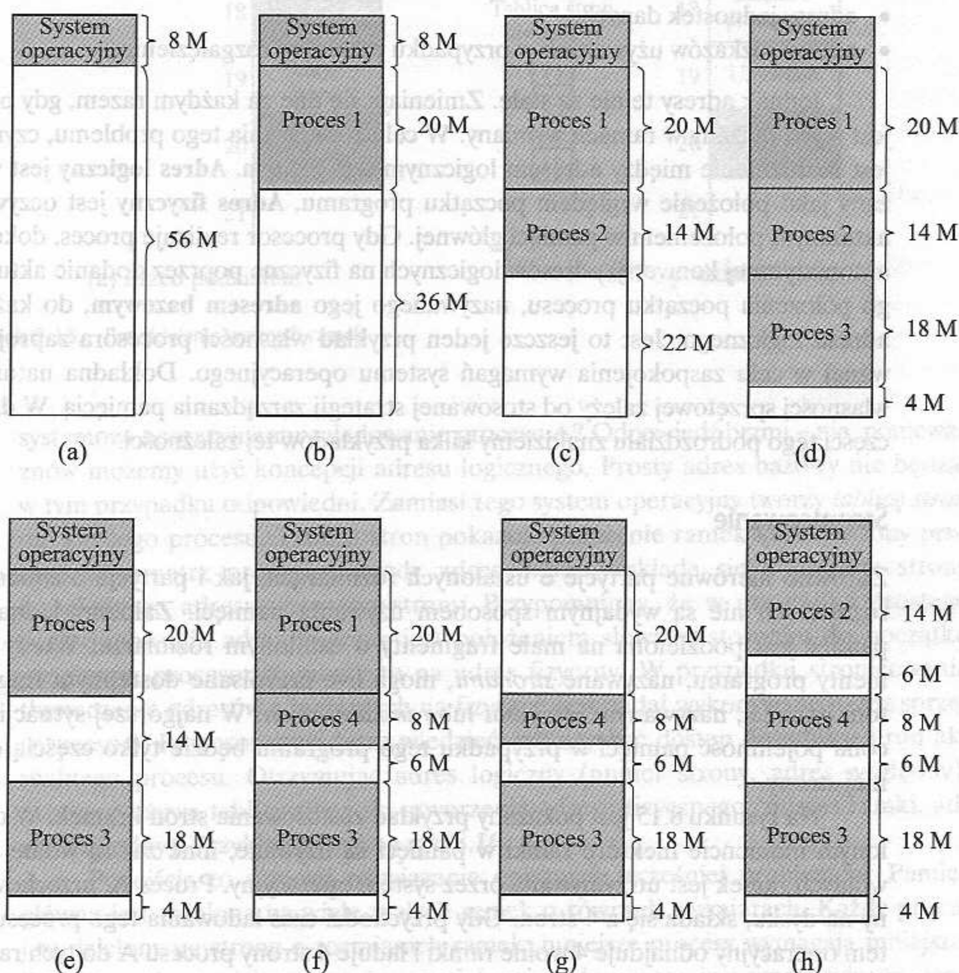
Nawet jeżeli używa się nierównych partycji o ustalonych rozmiarach, wystąpią straty pojemności pamięci. W większości przypadków proces nie będzie wymagał dokładnie takiej pojemności pamięci, jaką zapewnia partycja. Na przykład proces wymagający 3 MB pamięci zostanie umieszczony w partycji 4 MB (rys. 8.13b), co oznacza stratę 1 MB pojemności, która mogłaby być użyta przez inny proces.



Rysunek 8.13. Przykład partycjonowania pamięci 64 MB na bloki o ustalonym rozmiarze

Bardziej efektywnym podejściem jest użycie partycji o *zmiennych rozmiarach*. Gdy proces jest wprowadzany do pamięci, jest umieszczany w obszarze o dokładnie takiej pojemności, jaka jest niezbędna. Przykład takiego właśnie użycia pamięci 64 MB jest pokazany na rys. 8.14.

Początkowo pamięć główna jest pusta, z wyjątkiem części zajętej przez system operacyjny (a). Począwszy od miejsca, w którym kończy się system operacyjny ładowane są pierwsze trzy procesy, przy czym zajmują one tylko tyle miejsca, ile jest wymagane (b, c, d). Pozostaje luka na końcu pamięci, która jest za mała, aby zmieścić czwarty proces. W pewnej chwili żaden z procesów znajdujących się w pamięci nie jest w stanie gotowości. System operacyjny wymienia więc proces 2 (e), dzięki czemu powstaje miejsce wystarczające do załadowania nowego procesu, mianowicie procesu 4 (f). Ponieważ proces 4 jest mniejszy niż proces 2, powstaje następna mała



Rysunek 8.14. Rezultat partycjonowania dynamicznego

luka. Jak pokazuje ten przykład, metoda ta rozpoczyna się bardzo dobrze, jednak może prowadzić do sytuacji, w której występuje w pamięci wiele małych luk. W miarę upływu czasu pamięć jest coraz bardziej dzielona na części i wykorzystanie pamięci jest coraz mniej efektywne. Metodą pokonywania tego problemu jest *upakowanie* (*compaction*). Od czasu do czasu system operacyjny przesuwa procesy w pamięci w celu skumulowania wolnej pamięci w jednym bloku. Jest to procedura czasochłonna, powodująca stratę czasu procesora.

Zanim rozpatrzymy sposoby postępowania z wadami partycjonowania, musimy wyjaśnić jeszcze pewien problem. Jeśli czytelnik wróci na chwilę do rys. 8.14, to zauważy z pewnością, że najprawdopodobniej proces nie zostanie załadowany na to samo miejsce w pamięci głównej za każdym razem, gdy powraca w ramach wymiany. Proces w pamięci składa się z rozkazów i danych. Rozkazy zawierają adresy komórek pamięciowych dwóch rodzajów:

- adresy jednostek danych,
- adresy rozkazów używanych w przypadku rozkazów rozgałęzienia.

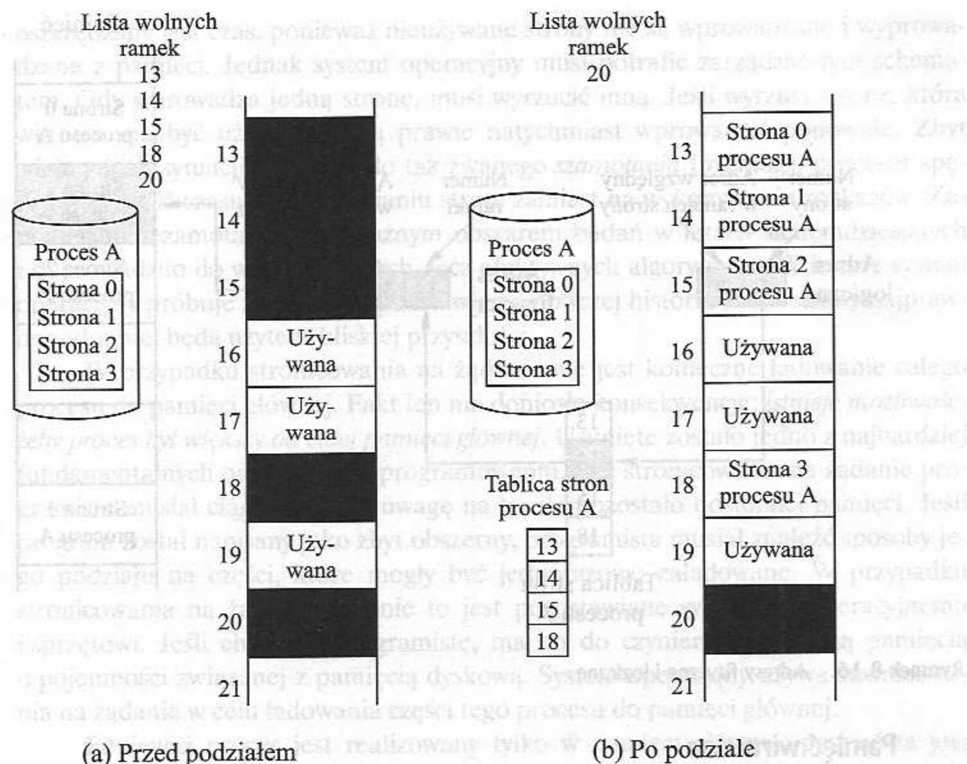
Jednak adresy te nie są stałe. Zmieniają się one za każdym razem, gdy proces jest wprowadzany w ramach wymiany. W celu rozwiązania tego problemu, czynione jest rozróżnienie między adresem logicznym a fizycznym. **Adres logiczny** jest wyrażany jako położenie względem początku programu. **Adres fizyczny** jest oczywiście aktualnym położeniem w pamięci głównej. Gdy procesor realizuje proces, dokonuje automatycznej konwersji adresów logicznych na fizyczne poprzez dodanie aktualnego położenia początku procesu, nazywanego jego **adresem bazowym**, do każdego adresu logicznego. Jest to jeszcze jeden przykład własności procesora zaprojektowanej w celu zaspokojenia wymagań systemu operacyjnego. Dokładna natura tej własności sprzętowej zależy od stosowanej strategii zarządzania pamięcią. W dalszej części tego podrozdziału znajdziemy kilka przykładów tej zależności.

Stronicowanie

Zarówno nierówne partycje o ustalonych rozmiarach, jak i partycje o zmiennych rozmiarach nie są wydajnym sposobem używania pamięci. Załóżmy jednak, że pamięć jest podzielona na małe fragmenty o ustalonym rozmiarze. Wtedy fragmenty programu, nazywane *stronami*, mogą być przypisane dostępnym fragmentom pamięci, nazywanym *ramkami* lub *ramkami stron*. W najgorszej sytuacji stracona pojemność pamięci w przypadku tego programu będzie tylko częścią ostatniej strony.

Na rysunku 8.15 jest pokazany przykład zastosowania stron i ramek. W określonym momencie niektóre ramki w pamięci są używane, inne zaś są wolne. Lista wolnych ramek jest utrzymywana przez system operacyjny. Proces A, przechowywany na dysku, składa się z 4 stron. Gdy przychodzi czas ładowania tego procesu, system operacyjny odnajduje 4 wolne ramki i ładuje 4 strony procesu A do tych ramek.

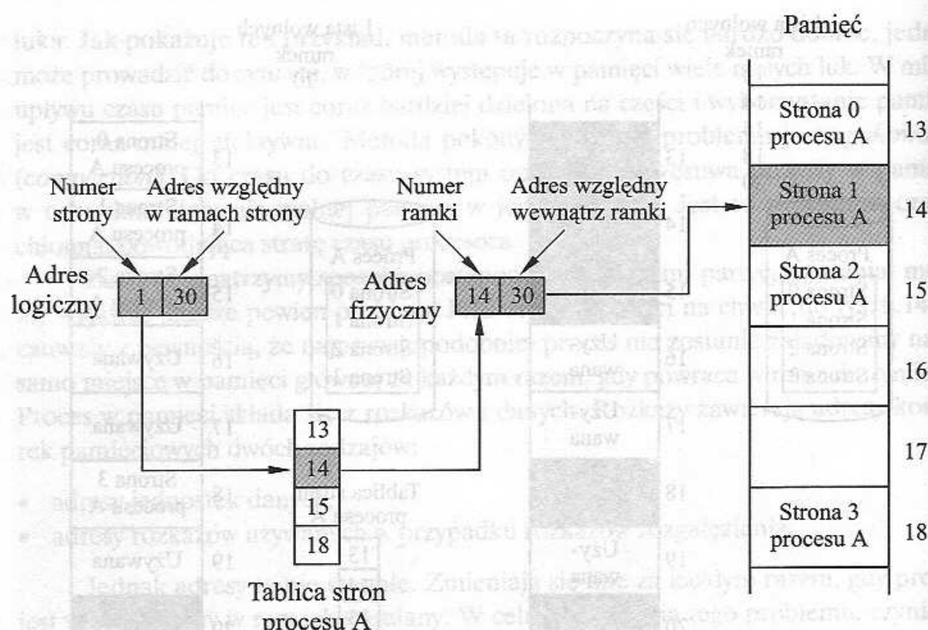
Załóżmy teraz, że, jak w tym przykładzie, nie ma wystarczającej liczby nieużywanych, sąsiadujących ze sobą ramek, aby zmieścić ten proces. Czy to uniemożliwi



Rysunek 8.15. Przydzielanie wolnych ramek

systemowi operacyjnemu załadowanie procesu A? Odpowiedź brzmi – nie, ponieważ znów możemy użyć koncepcji adresu logicznego. Prosty adres bazowy nie będzie w tym przypadku odpowiedni. Zamiast tego system operacyjny tworzy *tablicę stron* dla każdego procesu. Tablica stron pokazuje położenie ramek każdej strony procesu. Wewnątrz programu każdy adres logiczny składa się z numeru strony i względnego adresu wewnątrz strony. Przypomnijmy, że w przypadku prostego partycjonowania adres logiczny jest położeniem słowa w stosunku do początku programu; procesor tłumaczy to na adres fizyczny. W przypadku stronicowania tłumaczenie adresów z logicznych na fizyczne jest nadal wykonywane przez sprzęt (procesor). Procesor musi teraz wiedzieć, jak uzyskać dostęp do tablicy stron aktualnego procesu. Otrzymując adres logiczny (numer strony, adres względny), procesor używa tablicy stron do utworzenia adresu fizycznego (numer ramki, adres względny). Przykład widać na rys. 8.16.

Podejście to stanowi rozwiązanie opisanych wcześniej problemów. Pamięć główna jest dzielona na wiele małych ramek o równych rozmiarach. Każdy proces jest dzielony na strony o rozmiarach ramek: mniejsze procesy wymagają mniejszej liczby ramek, a większe procesy – większej. Gdy proces jest wprowadzany, jego strony są ładowane do dostępnych ramek i ustalana jest tablica stron.



Rysunek 8.16. Adresy fizyczne i logiczne

Pamięć wirtualna

Stronicowanie na żądanie

W wyniku zastosowania stronicowania stały się możliwe prawdziwie efektywne systemy wieloprogramowe. Równie ważne jest to, że prosta taktyka podziału procesu na strony doprowadziła do powstania innej ważnej koncepcji – pamięci wirtualnej.

Aby zrozumieć na czym polega pomysł pamięci wirtualnej, musimy dodać pewne ulepszenie do schematu stronicowania, który właśnie przedyskutowaliśmy. Ulepszeniem tym jest *stronicowanie na żądanie*, które oznacza po prostu, że każda strona procesu jest wprowadzana tylko wtedy, kiedy jest potrzebna, to znaczy na żądanie.

Rozważmy duży proces, składający się z długiego programu oraz z pewnej liczby zespołów danych. Przez pewien czas wykonywanie programu może być ograniczone do małej jego części (np. do podprogramu standardowego) i mogą być używane tylko jeden lub dwa zespoły danych. Jest to zgodne z zasadą lokalności, którą wprowadziliśmy w dodatku 4A do rozdz. 4. Byłoby oczywiście rozrzutnością ładowanie wielu stron procesu, gdy tylko kilka stron będzie używanych przed zawieszeniem programu. Lepiej wykorzystamy pamięć, ładując tylko kilka stron. Następnie, jeśli program rozgałęzi się do rozkazu znajdującego się na stronie poza pamięcią lub jeśli program odwoła się do danych znajdujących się na stronie poza pamięcią, zostanie zasygnalizowany *błąd strony*. Skłoni to system operacyjny do dostarczenia żądanej strony.

Tak więc w określonym momencie tylko kilka stron danego procesu znajduje się w pamięci i dzięki temu więcej procesów może pozostawać w pamięci. Ponadto

oszczędzany jest czas, ponieważ nieużywane strony nie są wprowadzane i wyprowadzane z pamięci. Jednak system operacyjny musi potrafić zarządzać tym schematem. Gdy wprowadza jedną stronę, musi wyrzucić inną. Jeśli wyrzuci stronę, która wkrótce ma być użyta, musi ją prawie natychmiast wprowadzić ponownie. Zbyt wiele takich sytuacji prowadzi do tak zwanego *szamotania* (*trashing*): procesor spędza większość czasu na wymienianiu stron, zamiast na wykonywaniu rozkazów. Zapobieganie szamotaniu było ważnym obszarem badań w latach siedemdziesiątych i doprowadziło do wielu złożonych, lecz efektywnych algorytmów. W istocie system operacyjny próbuje zgadnąć na podstawie najnowszej historii, które strony najprawdopodobniej będą użyte w bliskiej przyszłości.

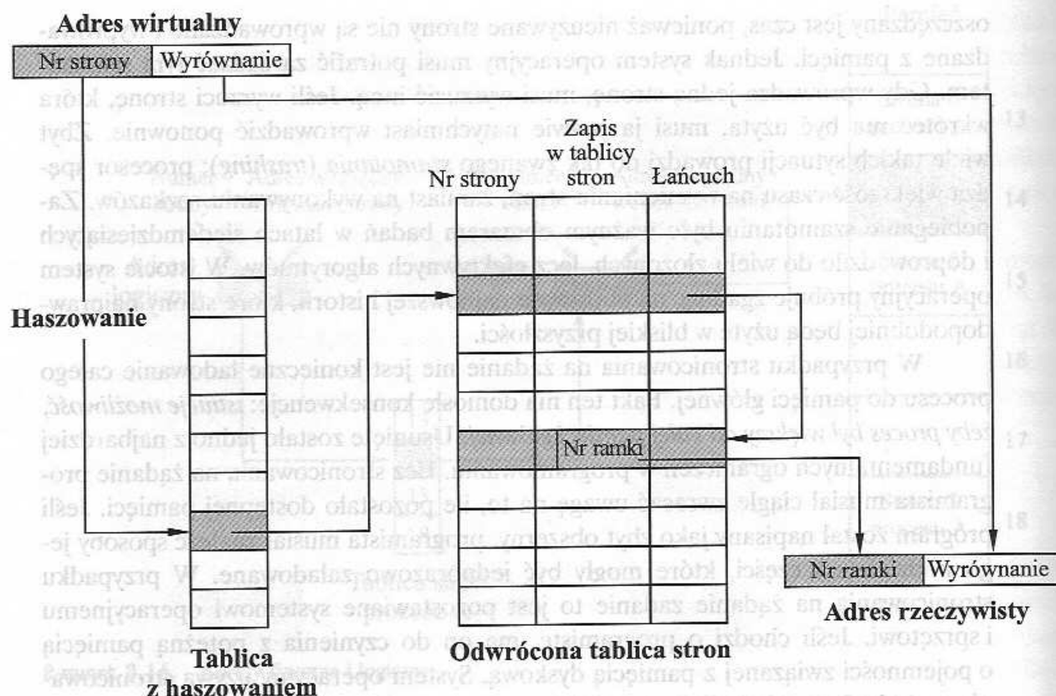
W przypadku stronicowania na żądanie nie jest konieczne ładowanie całego procesu do pamięci głównej. Fakt ten ma doniosłe konsekwencje: *istnieje możliwość, żeby proces był większy od całej pamięci głównej*. Usunięte zostało jedno z najbardziej fundamentalnych ograniczeń w programowaniu. Bez stronicowania na żądanie programista musiał ciągle zwracać uwagę na to, ile pozostało dostępnej pamięci. Jeśli program został napisany jako zbyt obszerny, programista musiał znaleźć sposoby jego podziału na części, które mogły być jednorazowo załadowane. W przypadku stronicowania na żądanie zadanie to jest pozostawiane systemowi operacyjnemu i sprzętowi. Jeśli chodzi o programistę, ma on do czynienia z potężną pamięcią o pojemności związanej z pamięcią dyskową. System operacyjny używa stronicowania na żądanie w celu ładowania części tego procesu do pamięci głównej.

Ponieważ proces jest realizowany tylko w pamięci głównej, pamięć ta jest określana jako **pamięć rzeczywista**. Jednak programista lub użytkownik dysponuje o wiele większą pamięcią – tą, która znajduje się na dysku. Ta ostatnia jest wobec tego nazywana **pamięcią wirtualną**. Pamięć wirtualna umożliwia bardzo efektywne wieloprogramowanie i uwalnia użytkownika od niepotrzebnie ciasnych ograniczeń pamięci głównej.

Struktura tablicy stron

Podstawowy mechanizm odczytu słowa z pamięci obejmuje przetłumaczenie adresu wirtualnego (lub logicznego), składającego się z numeru strony i adresu względnego, na adres fizyczny, składający się z numeru ramki i adresu względnego, za pomocą tablicy stron. Ponieważ tablica stron ma zmienną długość, zależną od rozmiaru procesu, nie możemy spodziewać się trzymania jej w rejestrach. Zamiast tego musi się ona znajdować w pamięci głównej. Na rysunku 8.16 jest pokazana propozycja rozwiązania sprzętowego tego schematu. Gdy jest realizowany określony proces, rejestr zachowuje adres początkowy tablicy stron tego procesu. Numer strony adresu wirtualnego służy do indeksowania tej tablicy oraz do znalezienia odpowiedniego numeru ramki. Po połączeniu go z adresem względnym, stanowiącym część adresu wirtualnego, można otrzymać żądany adres rzeczywisty.

W większości systemów występuje jedna tablica stron na proces. Jednak każdy proces może zajmować znaczną część pamięci wirtualnej. Na przykład w przypadku architektury VAX każdy proces może zajmować do 2^{31} = 2 GB pamięci wirtualnej.



Rysunek 8.17. Struktura odwróconej tablicy stron

Przy użyciu stron 2^9 = 512-bajtowych, oznacza to, że na jeden proces może przypadać 2^{22} zapisów tablicy stron. Jest jasne, że pojemność pamięci przeznaczona na same tylko tablice stron byłaby niedopuszczalnie duża. Aby uporać się z tym problemem, w większości przypadków tablice stron są przechowywane raczej w pamięci wirtualnej niż w rzeczywistej. Oznacza to, że tablice stron również podlegają stronicowaniu. Gdy proces jest realizowany, przynajmniej część jego tablicy stron musi się znajdować w pamięci głównej, w tym zapis tablicy stron odnoszący się do aktualnie realizowanej strony. Niektóre procesory stosują dwupoziomowy schemat organizacji dużych tablic stron. W takim przypadku występuje katalog stron, w którym każdy zapis oznacza tablicę stron. Jeśli długość katalogu stron wynosi X, a maksymalna długość tablicy stron jest równa Y, to liczba stron procesu może wynosić do $X \times Y$. Zwykle maksymalna długość tablicy stron jest ograniczona do jednej strony. Zobaczymy przykład takiego dwupoziomowego podejścia, gdy w dalszej części tego rozdziału będziemy rozważać Pentium II.

Alternatywnym rozwiązaniem w stosunku do stosowania jedno- i dwupoziomowych tablic stron jest użycie struktury *odwróconej tablicy stron* (rys. 8.17). Rozwiązanie to jest stosowane w komputerze AS/400 firmy IBM, a także we wszystkich wyrobach RISC tej firmy, łącznie z PowerPC.

W tym przypadku numer strony będący częścią adresu wirtualnego jest odwzorowywany w tablicy z haszowaniem (*hash table*) za pomocą prostej funkcji ha-

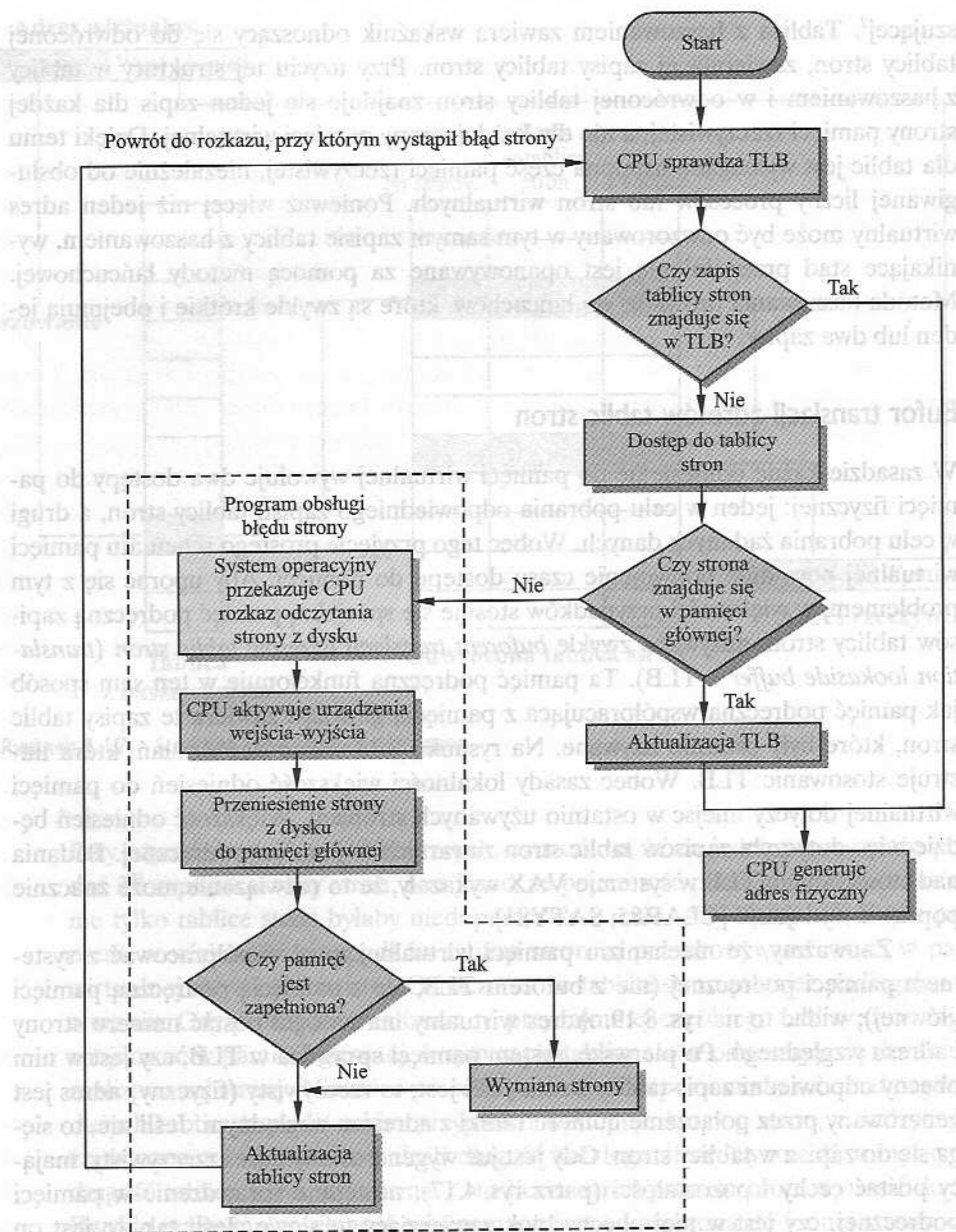
szującej[2]. Tablica z haszowaniem zawiera wskaźnik odnoszący się do odwróconej tablicy stron, zawierającej zapisy tablicy stron. Przy użyciu tej struktury w tablicy z haszowaniem i w odwróconej tablicy stron znajduje się jeden zapis dla każdej strony pamięci rzeczywistej, a nie dla każdej strony pamięci wirtualnej. Dzięki temu dla tablic jest wymagana ustalona część pamięci rzeczywistej, niezależnie od obsługiwanej liczby procesów lub stron wirtualnych. Ponieważ więcej niż jeden adres wirtualny może być odwzorowany w tym samym zapisie tablicy z haszowaniem, wynikające stąd przepełnienie jest opanowywane za pomocą metody łańcuchowej. Metoda haszowania prowadzi do łańcuchów, które są zwykle krótkie i obejmują jeden lub dwa zapisy.

Bufor translacji adresów tablic stron

W zasadzie każde odniesienie do pamięci wirtualnej wywołuje dwa dostępy do pamięci fizycznej: jeden w celu pobrania odpowiedniego zapisu tablicy stron, a drugi w celu pobrania żądanych danych. Wobec tego przyjęcie prostego schematu pamięci wirtualnej powoduje podwojenie czasu dostępu do pamięci. Aby uporać się z tym problemem, w większości przypadków stosuje się specjalną pamięć podręczną zapisów tablicy stron, nazywaną zwykle *buforem translacji adresów tablic stron* (*translation lookaside buffer* – TLB). Ta pamięć podręczna funkcjonuje w ten sam sposób jak pamięć podręczna współpracująca z pamięcią główną i zawiera te zapisy tablic stron, które były ostatnio używane. Na rysunku 8.18 widać sieć działań, która ilustruje stosowanie TLB. Wobec zasady lokalności większość odniesień do pamięci wirtualnej dotyczy miejsc w ostatnio używanych stronach. Większość odniesień będzie więc dotyczyła zapisów tablic stron zawartych w pamięci podręcznej. Badania nad stosowaniem TLB w systemie VAX wykazały, że to rozwiązanie może znacznie poprawić wydajność [CLAR85, SATY81].

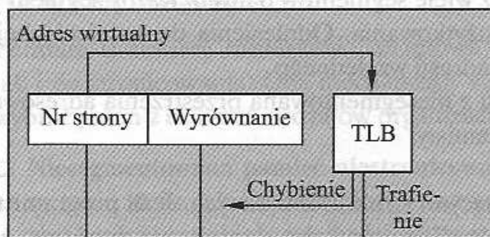
Zauważmy, że mechanizm pamięci wirtualnej musi współpracować z systemem pamięci podręcznej (nie z buforem TLB, ale z pamięcią podręczną pamięci głównej); widać to na rys. 8.19. Adres wirtualny ma na ogół postać numeru strony i adresu względnego. Po pierwsze, system pamięci sprawdza w TLB, czy jest w nim obecny odpowiedni zapis tablicy stron. Jeśli jest, to rzeczywisty (fizyczny) adres jest generowany przez połączenie numeru ramki z adresem względnym. Jeśli nie, to sięga się do zapisu w tablicy stron. Gdy jest już wygenerowany adres rzeczywisty, mający postać cechy i pozostałości (patrz rys. 4.17), następuje sprawdzenie w pamięci podręcznej, czy jest w niej obecny blok zawierający to słowo. Jeśli tak, to jest on zwracany do procesora. Jeśli nie, słowo jest pobierane z pamięci głównej.

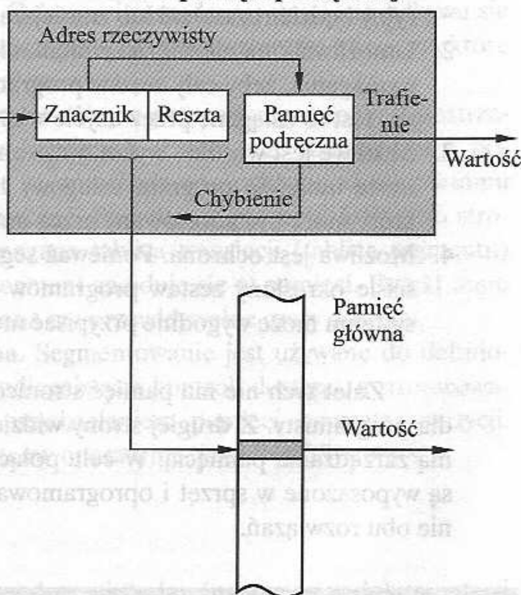
[2] Funkcja haszująca odwzorowuje liczby z zakresu 0 do M na liczby w zakresie 0 do N, gdzie $M > N$. Wynik funkcji haszującej jest używany jako indeks w tablicy z haszowaniem. Ponieważ więcej niż jedna liczba wejściowa daje taką samą liczbę wyjściową, liczba wejściowa może przy odwzorowywaniu natrafić w tablicy z haszowaniem zapis, który jest już zajęty. W takim przypadku nowa liczba musi być przeniesiona na inne miejsce tablicy z haszowaniem. Zwykle nowa liczba jest umieszczana w pierwszym najbliższym pustym obszarze, a do skojarzenia zapisów służy znacznik położenia początkowego. Bardziej szczegółowy opis tablic z haszowaniem zawiera [STAL01].



Rysunek 8.18. Funkcjonowanie stronicowania i bufora translacji adresów tablic stron (TLB) [FURH87]

Czytelnik z pewnością doceni złożoność sprzętową procesora wymaganą przy pojedynczym odniesieniu do pamięci. Adres wirtualny jest tłumaczony na rzeczywisty. Obejmuje to odniesienie do tablicy stron, która może się znajdować w TLB, w pamięci głównej lub na dysku. Poszukiwane słowo może być w pamięci podręcznej, w pamięci głównej lub na dysku. W tym ostatnim przypadku strona zawierająca





Rysunek 8.19. Działanie bufora translacji adresów tablic stron i pamięci podręcznej

słowo musi być załadowana do pamięci głównej, a jej blok musi trafić do pamięci podręcznej. Ponadto zapis tablicy stron odnoszący się do tej strony musi być zaktualizowany.

Segmentacja

Istnieje jeszcze inny sposób dzielenia adresowalnej pamięci, znany jako *segmentacja*. Podczas gdy stronicowanie jest niewidzialne dla programisty i dostarcza mu większej przestrzeni adresowej, segmentacja jest zwykle widzialna dla programisty i jest stosowana w celu wygodniejszego organizowania programów i danych oraz jako środek kojarzenia atrybutów przywileju i ochrony z rozkazami i danymi.

Segmentacja pozwala programiście widzieć pamięć jako składającą się z wielu przestrzeni adresowych lub segmentów. Segmenty mają zmienny, dynamiczny rozmiar. Zwykle programista lub system operacyjny przypisują programy i dane do różnych segmentów. Może być wiele segmentów programów przeznaczonych dla róż-

nych rodzajów programów i również wiele segmentów danych. Każdy segment może mieć przypisane prawa dostępu i użytkowania. Odniesienia do pamięci obejmują adres w formie numeru segmentu i adresu względnego.

Organizacja ta, w porównaniu z niesegmentowaną przestrzenią adresową, ma wiele zalet z punktu widzenia programisty:

1. Upraszcza ona operowanie rosnącymi strukturami danych. Jeśli programista nie wie z góry, jak duża stanie się określona struktura danych, nie musi tego zgadywać. Struktura danych może mieć przypisany własny segment, a system operacyjny będzie rozszerzał lub zmniejszał segment zależnie od potrzeby.
2. Umożliwia ona niezależne zmienianie i ponowne kompilowanie programów, bez wymagania, żeby cały zestaw programów był ponownie konsolidowany i ładowany. Jest to osiągane przez użycie wielu segmentów.
3. Możliwe jest wspólne wykorzystywanie danych przez różne procesy. Programista może umieścić program usługowy lub użyteczną tablicę danych w segmencie, który może być adresowany przez inne procesy.
4. Możliwa jest ochrona. Ponieważ segment może być utworzony jako zawierający ściśle określony zestaw programów lub danych, programista lub administrator systemu może wygodnie przypisać mu prawa dostępu.

Zalet tych nie ma pamięć stronicowana, gdyż stronicowanie jest niewidzialne dla programisty. Z drugiej strony widzieliśmy, że stronicowanie jest efektywną formą zarządzania pamięcią. W celu połączenia zalet obu rozwiązań niektóre systemy są wyposażone w sprzęt i oprogramowanie systemowe, które umożliwiają stosowanie obu rozwiązań.

8.4. Zarządzanie pamięcią w Pentium II i PowerPC

Rozwiązanie sprzętowe zarządzania pamięcią w Pentium II

Od czasu wprowadzenia architektury 32-bitowej zostały opracowane wyrafinowane schematy zarządzania pamięcią w systemach mikroprocesorowych, przy czym wykorzystano doświadczenia nagromadzone w odniesieniu do systemów średnich i dużych. W wielu przypadkach wersje mikroprocesorowe są lepsze od wielkosystemowych poprzedników. Ponieważ schematy te były opracowywane przez producentów sprzętu mikroprocesorowego i mogą współpracować z wieloma systemami operacyjnymi, są raczej schematami o ogólnym przeznaczeniu. Reprezentatywnym przykładem jest schemat użyty w Pentium II. Sprzętowa strona zarządzania pamięcią Pentium II jest zasadniczo taka sama, jak w przypadku procesorów 80386 i 80486, chociaż występują pewne ulepszenia.

Przestrzenie adresowe

Pentium II zawiera rozwiązania sprzętowe zarówno wymagane do segmentowania, jak i do stronicowania. Oba mechanizmy są blokowane, co pozwala użytkownikowi wybrać jeden z czterech rodzajów organizacji pamięci:

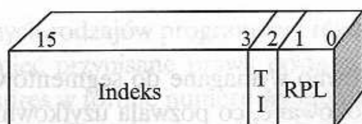
- **Niesegmentowana pamięć niestronicowana.** W tym przypadku adresy wirtualne są takie same jak fizyczne. Jest to użyteczne w przypadku zastosowań o małej złożoności i wysokiej wydajności, np. w sterownikach.
- **Niesegmentowana pamięć stronicowana.** Pamięć jest tu widziana jako stronicowana, liniowa przestrzeń adresowa. Ochrona i zarządzanie pamięcią odbywa się poprzez stronicowanie. Takie rozwiązanie jest faworyzowane przez niektóre systemy operacyjne, np. UNIX Berkeley.
- **Segmentowana pamięć niestronicowana.** W tym przypadku pamięć jest postrzegana jako zbiór przestrzeni adresów logicznych. Zaletą tego rozwiązania w porównaniu ze stronicowaniem jest to, że umożliwia ono ochronę aż do poziomu pojedynczego bajta, jeśli jest to konieczne. Ponadto, w przeciwieństwie do stronicowania, gwarantuje ono, że wymagana tablica translacji (tablica segmentu) znajduje się w mikroukładzie, gdy segment znajduje się w pamięci. Dzięki temu pamięć segmentowana niestronicowana ma przewidywalne czasy dostępu.
- **Segmentowana pamięć stronicowana.** Segmentowanie jest używane do definiowania logicznych partycji pamięci podlegających kontroli dostępu, a stronicowanie jest używane do zarządzania przydzielaniem pamięci wewnątrz partycji. Rozwiązanie to faworyzują takie systemy operacyjne, jak UNIX System V.

Segmentacja

Gdy stosowana jest segmentacja, każdy adres wirtualny (nazywany w dokumentacji Pentium II adresem logicznym) składa się z 16-bitowego odniesienia do segmentu i 32-bitowego adresu względnego. Dwa bity odniesienia do segmentu służą mechanizmowi ochrony, pozostaje więc 14 bitów na określenie segmentu. Wobec tego w przypadku pamięci niesegmentowanej całkowita wirtualna przestrzeń adresowa widziana przez użytkownika wynosi $2^{32} = 4$ GB. W przypadku pamięci segmentowanej wynosi ona $2^{46} = 64$ TB (terabajtów). W przestrzeni adresów fizycznych są stosowane adresy 32-bitowe, co daje maksymalnie 4 GB.

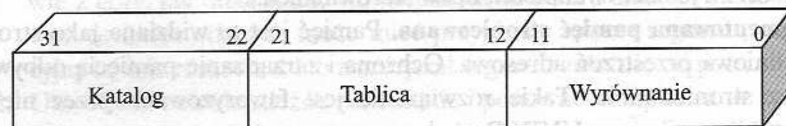
W rzeczywistości pojemność pamięci wirtualnej może być większa niż 64 TB. Jest tak, ponieważ interpretacja adresu wirtualnego przez procesor zależy od tego, który proces jest aktualnie czynny. Przestrzeń adresu wirtualnego jest dzielona na dwie części. Połowa przestrzeni adresu wirtualnego (8 K segmentów × 4 GB) jest globalna, dostępna dla wszystkich procesów; pozostała część ma charakter lokalny i jest odrębna dla każdego procesu.

Z każdym segmentem są związane dwie formy ochrony: poziom uprzywilejowania i atrybut dostępu. Występują cztery poziomy uprzywilejowania, od najbardziej chronionego (poziom 0) do najmniej chronionego (poziom 3). Poziom uprzywilejowania związany z segmentem danych jest jego „poziomem poufności" (*classification*);

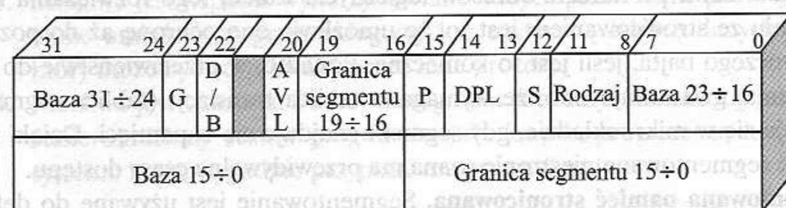


TI – wskaźnik tablicy
RPL – poziom uprzywilejowania

(a) Selektor segmentu



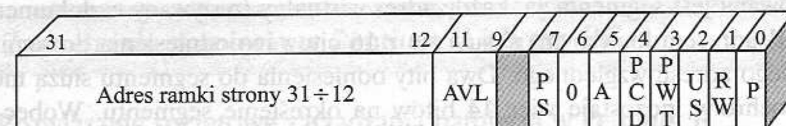
(b) Adres liniowy



AVL – dostępne dla oprogramowania systemowego
Baza – adres segmentu bazowego
D/B – domyślny rozmiar operacji
DPL – rozmiar deskryptora uprzywilejowania
▒ – zarezerwowany

G – stopień granulacji
P – obecność segmentu
Rodzaj – rodzaj segmentu
S – rodzaj deskryptora

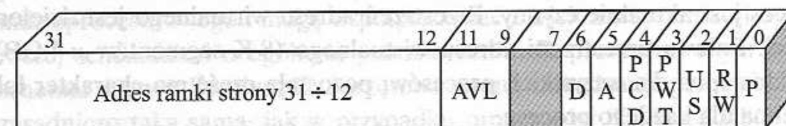
(c) Deskryptor segmentu (zapis tablicy segmentów)



AVL – dostępne dla programisty systemów
PS – rozmiar strony
A – potwierdzenie dostępu do strony
PCD – blokada pamięci podręcznej

PWT – jednoczesny zapis
US – użytkownik/nadzorca
RW – odczyt/zapis
P – obecna

(d) Zapis katalogu stron



D – potwierdzenie zapisu

(e) Zapis tablicy stron

Rysunek 8.20. Formaty zarządzania pamięcią procesora Pentium

poziom uprzywilejowania związany z segmentem programu jest jego „poziomem dostępności" (*clearance*). Realizowany program może mieć dostęp tylko do takich segmentów danych, w stosunku do których jego poziom dostępności jest niższy (tzn. bardziej uprzywilejowany) lub równy (tzn. tak samo uprzywilejowany) poziomowi uprzywilejowania segmentu danych.

Sprzęt nie dyktuje sposobu używania tych poziomów uprzywilejowania; zależy to od projektu i implementacji systemu operacyjnego. Intencją projektantów było, aby poziom uprzywilejowania 1 był używany przez większą część kodu systemu operacyjnego, zaś poziom 0 – przez niewielką część systemu operacyjnego przeznaczoną do zarządzania, ochrony i kontroli dostępu do pamięci. Dwa poziomy pozostają aplikacjom. W wielu systemach programy użytkowe pozostają na poziomie 3, przy czym poziom 2 jest nieużywany. Wyspecjalizowane podsystemy programów użytkowych, które muszą być chronione, ponieważ stosują własne mechanizmy zabezpieczeń, mogą kandydować do poziomu 2. Przykładami są systemy baz danych, systemy automatyzacji urzędów i środowiska inżynierii oprogramowania.

Oprócz kontrolowania dostępu do segmentów danych mechanizm uprzywilejowania ogranicza stosowanie pewnych rozkazów. Niektóre rozkazy, jak np. rozkazy operujące rejestrami zarządzania pamięcią, mogą być wykonywane wyłącznie na poziomie 0. Rozkazy wejścia-wyjścia mogą być wykonywane wyłącznie do pewnego poziomu określanego przez system operacyjny; zwykle jest to poziom 1.

Atrybut dostępu segmentu danych określa, czy dozwolone są operacje odczytu i zapisu, czy tylko odczytu. W przypadku segmentów programu atrybut dostępu określa, czy dozwolone jest odczytywanie i wykonywanie, czy tylko odczytywanie.

W przypadku segmentowania mechanizm translacji adresu obejmuje odwzorowywanie adresu wirtualnego na tzw. adres liniowy (rys. 8.20b). Adres wirtualny składa się z 32-bitowego adresu względnego i z 16-bitowego selektora segmentu (rys. 8.20a). Selektor segmentu składa się z trzech pól. Są to:

- **Wskaźnik tablicy** (*table indicator* – TI). Wskazuje, czy do translacji powinna być używana tablica segmentu ogólnego, czy tablica segmentu lokalnego.
- **Numer segmentu.** Określa numer segmentu służący jako indeks w tablicy segmentu.
- **Wymagany poziom uprzywilejowania** (*requested privilege level* – RPL). Poziom uprzywilejowania wymagany dla tego dostępu.

Każdy zapis w tablicy segmentu obejmuje 64 bity, co widać na rys. 8.20c. Pola są zdefiniowane w tabeli 8.5.

Stronicowanie

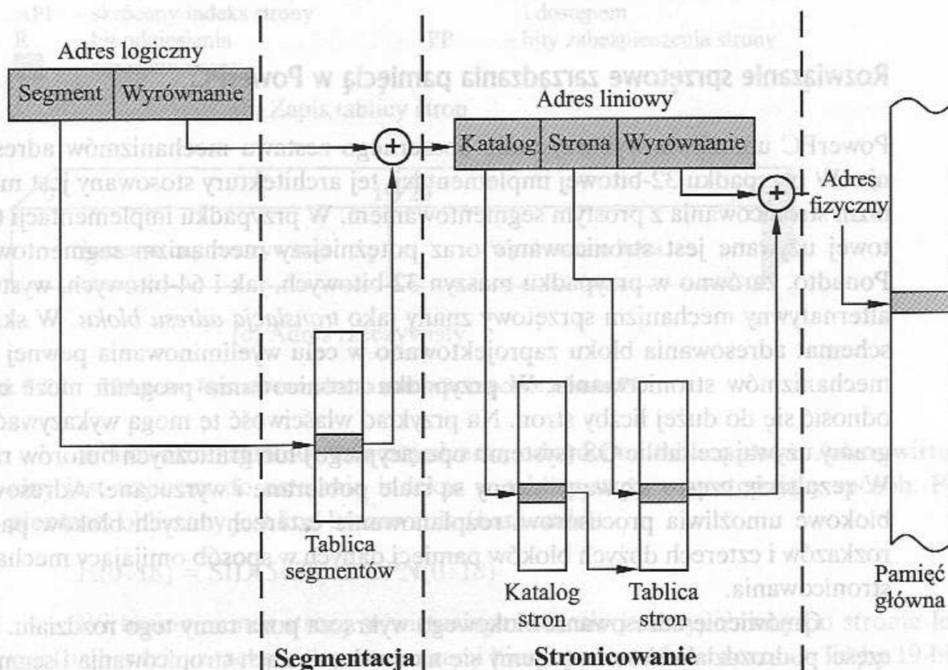
Segmentowanie jest cechą opcjonalną i może być zablokowane. Gdy segmentowanie jest stosowane, adresy używane w programach są adresami wirtualnymi i są przekształcane na adresy liniowe w sposób uprzednio opisany. Gdy segmentowanie nie jest stosowane, w programach są używane adresy liniowe. W każdym przypadku potrzebne jest następujące działanie w celu przekształcenia adresu liniowego na rzeczywisty adres 32-bitowy.

Tabela 8.5. Parametry zarządzania pamięcią procesora Pentium II

Deskryptor segmentu (zapis tablicy segmentu)
Baza
Określa adres początkowy segmentu wewnątrz 4-gigabajtowej liniowej przestrzeni adresowej.
Bit D/B
W segmencie programu jest to bit D, który wskazuje, czy argumenty i tryby adresowania są 16-, czy 32-bitowe.
Poziom uprzywilejowania deskryptora (DPL)
Określa poziom uprzywilejowania segmentu, do którego odnosi się ten deskryptor.
Bit granulacji (G)
Wskazuje, czy pole Granica ma być interpretowane w jednostkach 1 bajta, czy też 4 KB.
Granica
Definiuje rozmiar segmentu. Procesor interpretuje pole Granica na jeden z dwóch sposobów, zależnie od bitu granulacji: w jednostkach 1 bajta, aż do granicy rozmiaru segmentu równej 1 MB, lub w jednostkach 4 KB, aż do granicy 4 GB.
Bit S
Określa, czy dany segment jest segmentem systemowym, programów czy danych.
Bit obecności segmentu (P)
Używany w przypadku systemów niestronicowanych. Wskazuje, czy segment jest obecny w pamięci głównej. W przypadku systemów stronicowanych bit ten jest zawsze ustawiany na 1.
Rodzaj
Umożliwia rozróżnienie różnych rodzajów segmentów i wskazanie atrybutów dostępu.
Zapisy katalogu stron i tablicy stron
Bit dostępu (A)
Bit ten jest ustawiany na wartość 1 przez procesor na obu poziomach tablicy strony, gdy ma miejsce operacja odczytu lub zapisu w odpowiedniej stronie.
Bit zapisu (D)
Bit ten jest ustawiany na 1 przez procesor, gdy ma miejsce zapis na odpowiedniej stronie.
Adres ramki strony
Określa fizyczny adres strony w pamięci, jeśli jest ustawiony bit obecności. Ponieważ ramki stron są wyrównane z granicami 4 K, dolnymi 12 bitami są 0, a tylko górne 20 bitów włączono do zapisu. W katalogu stron jest to adres tablicy stron.
Bit blokady strony (PCD)
Wskazuje, czy dane zawarte na stronie mogą być umieszczane w pamięci podręcznej.
Bit rozmiaru strony (PS)
Wskazuje, czy rozmiar strony jest równy 4 KB, czy 4 MB.
Bit zapisu jednoczesnego strony (PWT)
Wskazuje, czy przy zapisywaniu danych na odpowiedniej stronie będzie stosowany zapis jednoczesny, czy też zapis opóźniony.
Bit obecności (P)
Wskazuje, czy tablica strony lub strona znajduje się w pamięci głównej.
Bit zapisu-odczytu (RW)
W przypadku stron z poziomu użytkownika wskazuje, czy strona może być tylko odczytywana, czy też również zapisywana przy wykonywaniu programów z poziomu użytkownika.
Bit użytkownik-nadzorca (US)
Wskazuje, czy strona jest dostępna tylko dla systemu operacyjnego (poziom nadzorcy), czy też jest również dostępna dla aplikacji (poziom użytkownika).

Aby zrozumieć strukturę adresu liniowego, trzeba wiedzieć, że mechanizm stronicowania procesora Pentium II jest w istocie dwupoziomową operacją przekształcania tablicowego. Pierwszy poziom stanowi katalog stron, zawierający do 1024 zapisów. Dzieli to 4-gigabajtową, liniową przestrzeń pamięci na 1024 grupy stron, z których każda ma własną tablicę stron i każda ma długość 4 MB. Każda tablica stron zawiera do 1024 zapisów; każdy zapis odpowiada pojedynczej stronie 4-kilobajtowej. Zarządzanie pamięcią dysponuje opcjami używania jednego katalogu stron dla wszystkich procesów, jednego katalogu stron dla każdego procesu lub pewnej kombinacji tych rozwiązań. Katalog stron odnoszący się do zadań bieżących znajduje się zawsze w pamięci głównej. Tablice stron mogą pozostawać w pamięci wirtualnej.

Na rysunku 8.20 są pokazane formaty zapisów w katalogach stron i w tablicach stron, natomiast pola są zdefiniowane w tabeli 8.5. Zauważmy, że mechanizmy kontroli dostępu mogą być wprowadzane przy użyciu stron lub grup stron.



Rysunek 8.21. Mechanizmy translacji adresu w pamięci procesora Pentium

W Pentium II jest również wykorzystywany bufor translacji adresów tablic stron. Bufor może zawierać 32 zapisy tablic stron. Za każdym razem, gdy zmieniany jest katalog stron, bufor jest kasowany.

Na rysunku 8.21 jest pokazana kombinacja mechanizmów segmentowania i stronicowania. Dla jasności pominięto bufor translacji adresów tablic stron oraz pamięć podręczną.

Pentium II zawiera wreszcie nową możliwość, nie występującą w 80386 ani 80486 – możliwość dwóch rozmiarów stron. Jeśli bit PSE (*page size extension* – rozszerzenie rozmiaru strony) w rejestrze sterowania 4 jest ustawiony na 1, to jednostka stronicowania pozwala programiście systemu operacyjnego na określenie rozmiaru strony albo jako 4 KB, albo 4 MB.

Gdy stosowane są strony 4-megabajtowe, występuje tylko jeden poziom przekształcania tablicowego. Jeśli ma miejsce dostęp sprzętu do katalogu stron, to bit PS w katalogu stron (rys. 8.20d) jest ustawiany na 1. W tym przypadku bity od 9 do 21 są ignorowane, a bity od 22 do 31 określają adres bazowy 4-megabajtowej strony w pamięci. Występuje więc tylko jedna tablica stron.

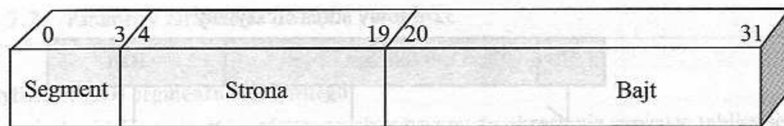
Stosowanie stron 4-megabajtowych zmniejsza wymagania odnoszące się do zawartości związanej z zarządzaniem pamięcią w dużych pamięciach głównych. W przypadku stron 4-kilobajtowych, pełna pamięć główna o pojemności 4 GB wymaga około 4 MB na same tylko tablice stron. W przypadku stron 4-megabajtowych do zarządzania pamięcią wystarcza jedna tablica o długości 4 KB.

Rozwiązanie sprzętowe zarządzania pamięcią w PowerPC

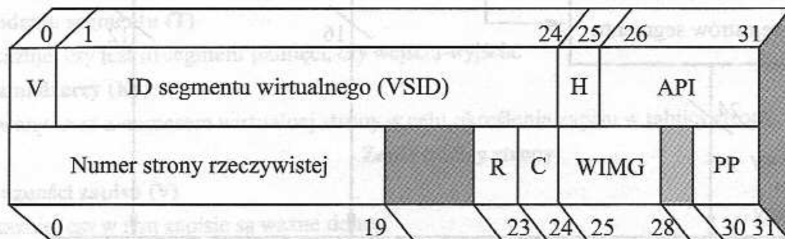
PowerPC umożliwia wykorzystanie obszernego zestawu mechanizmów adresowania. W przypadku 32-bitowej implementacji tej architektury stosowany jest mechanizm stronicowania z prostym segmentowaniem. W przypadku implementacji 64-bitowej używane jest stronicowanie oraz potężniejszy mechanizm segmentowania. Ponadto, zarówno w przypadku maszyn 32-bitowych, jak i 64-bitowych, występuje alternatywny mechanizm sprzętowy znany jako *translacja adresu bloku*. W skrócie, schemat adresowania bloku zaprojektowano w celu wyeliminowania pewnej wady mechanizmów stronicowania. W przypadku stronicowania program może często odnosić się do dużej liczby stron. Na przykład właściwość tę mogą wykazywać programy używające tablic OS (systemu operacyjnego) lub graficznych buforów ramki. W rezultacie często używane strony są stale pobierane i wyrzucane. Adresowanie blokowe umożliwia procesorowi rozplanowanie czterech dużych bloków pamięci rozkazów i czterech dużych bloków pamięci danych w sposób omijający mechanizm stronicowania.

Omówienie adresowania blokowego wykracza poza ramy tego rozdziału. W tej części podrozdziału skoncentrujemy się na mechanizmach stronicowania i segmentowania PowerPC. Schemat 64-bitowy jest podobny.

W 32-bitowym PowerPC jest używany 32-bitowy adres efektywny (rys. 8.22a). Adres ten zawiera 16-bitowy identyfikator strony i 12-bitowy selektor bajta. Wobec tego możliwe jest używanie $2^{12} = 4$ KB stron. Dopuszcza się do $2^{16} = 64$ K stron na segment. Cztery bity adresu są używane do wyznaczania jednego z 16 rejestrów segmentu. Zawartość tych rejestrów jest kontrolowana przez system operacyjny. Każdy rejestr segmentu zawiera bity kontroli dostępu oraz 24-bitowy identyfikator, dzięki czemu 32-bitowy adres efektywny jest odwzorowywany na 52-bitowy adres wirtualny (rys. 8.23).

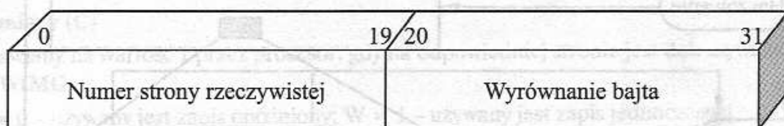


(a) Adres efektywny



V – bit ważności zapisu
H – identyfikator funkcji haszującej
API – skrócony indeks strony
R – bit odniesienia
▒ – zarezerwowany
C – bit zmiany
WIMG – bity sterowania pamięcią podręczną i dostępem
PP – bity zabezpieczenia strony

(b) Zapis tablicy stron



(c) Adres rzeczywisty

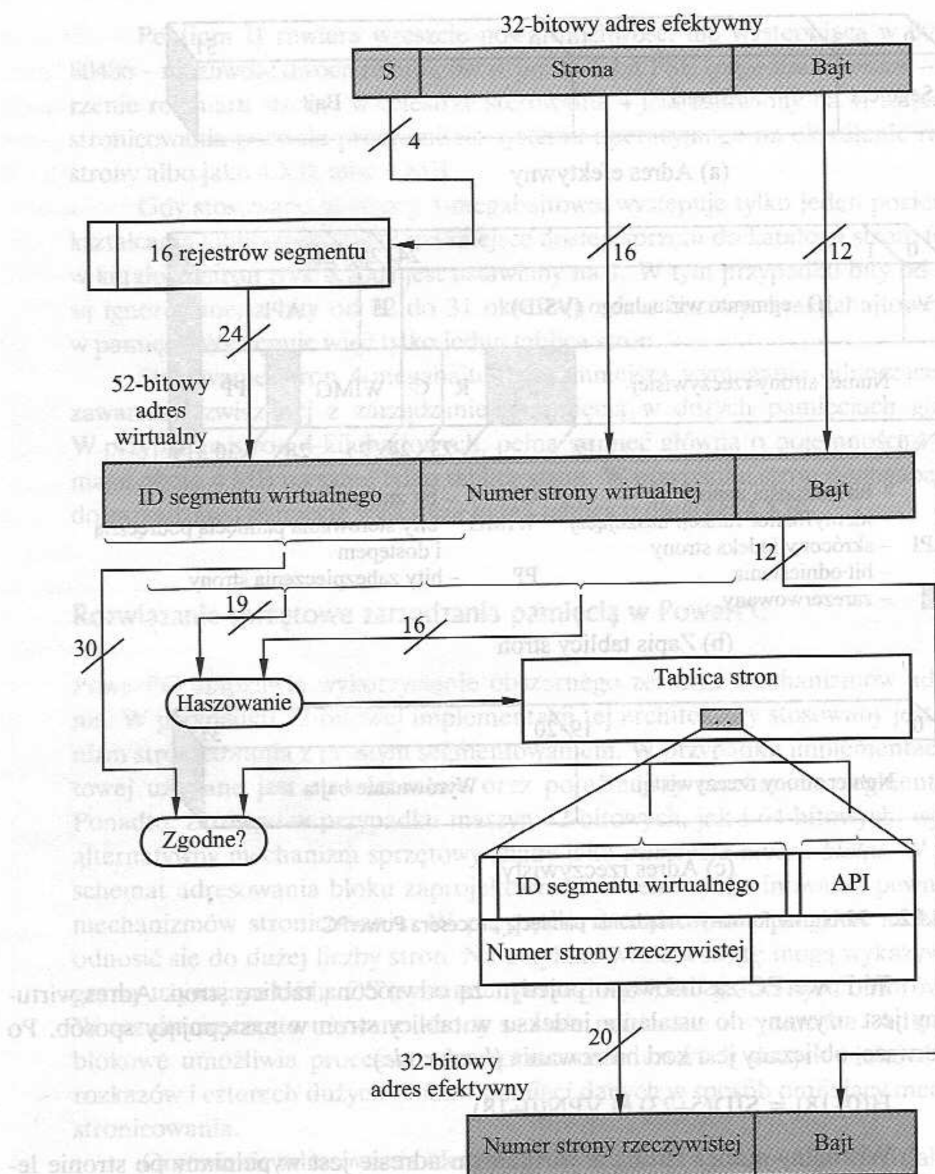
Rysunek 8.22. 32-bitowe formaty zarządzania pamięcią procesora PowerPC

W PowerPC zastosowano pojedynczą odwróconą tablicę stron. Adres wirtualny jest używany do ustalania indeksu w tablicy stron w następujący sposób. Po pierwsze, obliczany jest kod haszowania (*hash code*):

$$H(0 \div 18) = SID(5 \div 23) \oplus VPN(0 \div 18)$$

Wirtualny numer strony w wirtualnym adresie jest wypełniany po stronie lewej (najbardziej znaczącej) trzema zerami binarnymi w celu utworzenia liczby 19-bitowej. Następnie, bit po bicie obliczane jest LUB wykluczające tej liczby oraz 19 najmniej znaczących bitów identyfikatora wirtualnego segmentu w celu utworzenia 19-bitowego kodu haszowania. Tablica jest organizowana jako *n* grup 8 zapisów. Od 10 do 19 bitów kodu haszowania (zależnie od rozmiaru tablicy stron) jest używanych do wybierania jednej z grup tablicy. Następnie sprzęt zarządzania pamięcią przegląda 8 zapisów grupy w celu sprawdzenia zgodności z adresem wirtualnym.

Aby umożliwić sprawdzanie zgodności, każdy zapis tablicy stron zawiera identyfikator segmentu wirtualnego i 6 najbardziej znaczących bitów wirtualnego numeru strony nazywanych *skróconym indeksem strony* (ponieważ przynajmniej 10 bitów



Rysunek 8.23. Translacja 32-bitowego adresu procesora PowerPC

z 16-bitowego wirtualnego numeru strony zawsze uczestniczy w haszowaniu w celu wybrania grupy zapisów tablicy stron, tylko skrócona forma wirtualnego numeru strony musi być przeniesiona do zapisu tablicy strony, aby zapewnić jednoznaczną zgodność z adresem wirtualnym). Jeśli występuje zgodność, to 20-bitowy rzeczywisty numer strony z adresu jest wiązany z niższymi 12 bitami adresu efektywnego w celu utworzenia 32-bitowego adresu fizycznego, do którego wystąpiło odwołanie.

Tabela 7.7. Parametry zarządzania pamięcią PowerPC

Zapis tablicy segmentów
Identyfikator (ID) segmentu efektywnego
Wskazuje 1 z 64 G segmentów efektywnych; używany do określania zapisu w tablicy segmentu.
Bit ważności zapisu (V)
Wskazuje, czy jest to segment pamięci, czy wejścia-wyjścia.
Bit rodzaju segmentu (T)
Wskazuje, czy jest to segment pamięci, czy wejścia-wyjścia.
Klucz nadzorcy (Ks)
Używany wraz z numerem wirtualnej strony w celu określenia zapisu w tablicy strony.
Zapis tablicy strony
Bit ważności zapisu (V)
Wskazuje, czy w tym zapisie są ważne dane.
Identyfikator funkcji haszującej (H)
Wskazuje, czy jest to zapis pierwotny, czy wtórny z haszowaniem.
Skrócony wskaźnik strony (API)
Używany do jednoznacznego dopasowania adresu wirtualnego.
Bit odniesienia (R)
Ustawiany na wartość 1 przez procesor, gdy na odpowiedniej stronie ma miejsce operacja odczytu lub zapisu.
Bit zmiany (C)
Ustawiany na wartość 1 przez procesor, gdy na odpowiedniej stronie jest dokonywany zapis.
Bity WIMG
W = 0 – używany jest zapis opóźniony; W = 1 – używany jest zapis jednoczesny; I = 0 – przenoszenie do pamięci podręcznej nie jest zabronione; I = 1 – przenoszenie do pamięci podręcznej zabronione; M = 0 – pamięć nie jest wspólna; M = 1 – pamięć wspólna; G = 0 – pamięć niechroniona; G = 1 – pamięć chroniona.
Bity zabezpieczenia strony (PP)
Bity kontroli dostępu używane wraz z bitami K zapisu tablicy segmentu lub rejestru segmentu w celu określenia praw dostępu.

Jeśli zgodność nie występuje, to jest uzupełniany kod haszowania w celu utworzenia nowego indeksu tablicy stron, znajdującego się w takiej samej pozycji względnej na przeciwnym końcu tablicy. Grupa ta jest następnie przeglądana w celu zbadania zgodności. Jeśli zgodność nie zostanie stwierdzona, następuje przerwanie z powodu błędu strony.

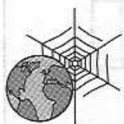
Na rysunku 8.22 widać schemat logiczny mechanizmu translacji adresu, a na rys. 8.23 są pokazane formaty efektywnego adresu, zapisu tablicy stron i adresu rzeczywistego. W tabeli 8.6 są zdefiniowane parametry zapisu tablicy stron.

Schemat zarządzania pamięcią odnoszący się do 64-bitowego PowerPC został tak zaprojektowany, aby był kompatybilny (jednokierunkowo) z implementacją 32-bitową. W zasadzie wszystkie adresy efektywne, rejestry ogólne i rejestry adresu skoku zostały rozszerzone w lewo do 64 bitów.

8.5. Polecana literatura i witryny WWW

Szczegółowe omówienie zagadnień przedstawionych w tym rozdziale znajduje się w [STAL01].

STAL01 Stallings W.: *Operating Systems, Internals and Design Principles*, 4th edition. Upper Saddle River, Prentice Hall, 2001.



Polecane witryny WWW:

- **Operating System Project Information.** Łącza do przedsięwzięć i prac badawczych dotyczących systemów operacyjnych.
- **ACM Special Interest Group on Operating System.** Informacje o publikacjach i konferencjach SIGOPS.
- **IEEE Technical Committee on Operating Systems and Applications.** Zawiera biuletyn on-line i łącza do innych witryn.
- **Review of Operating Systems.** Obszerny przegląd systemów operacyjnych komercyjnych, bezpłatnych, badawczych i amatorskich.

8.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Adres fizyczny – *physical address*
Adres logiczny – *logical address*
Blok sterowania procesami – *process control block*
Bufor translacji adresów stron pamięci (TLB) – *translation lookaside buffer*
Instrukcja uprzywilejowana – *privileged instruction*
Jądro – *kernel*
Język sterowania zadaniami (JCL) – *job control language*
Konwersacyjny system operacyjny – *interactive operating system*
Ochrona pamięci – *memory protection*
Pamięć rzeczywista – *real memory*
Pamięć wirtualna – *virtual memory*
Partycjonowanie – *partitioning*
Proces – *process*
Program narzędziowy – *utility*
Przerwanie – *interrupt*
Rezydentny program zarządzający – *resident monitor*
Segmentacja – *segmentation*
Stan procesu – *process state*
Stronicowanie – *paging*
Stronicowanie na żądanie – *demand paging*
System operacyjny (OS) – *operating system*
System podziału czasu – *time-sharing system*
System wsadowy – *batch system*
Szamotanie – *thrashing*
Szeregowanie długookresowe – *long-term scheduling*
Szeregowanie krótkookresowe – *short-term scheduling*
Szeregowanie średniookresowe – *medium-term scheduling*
Tablica stron – *page table*
Wieloprogramowanie – *multiprogramming*
Wielozadaniowość – *multitasking*
Wymiana – *swapping*
Zarządzanie pamięcią – *memory management*

Pytania kontrolne

8.1. Czym jest system operacyjny?

8.2. Wymień i krótko zdefiniuj podstawowe usługi realizowane przez system operacyjny.

8.3. Wymień i krótko zdefiniuj podstawowe rodzaje szeregowania pracy systemu operacyjnego.

8.4. Jaka jest różnica między procesem a programem?

8.5. Jaki jest cel wymiany?

8.6. Jeśli proces może być dynamicznie przypisywany różnym lokacjom w pamięci głównej, jakie są tego implikacje w odniesieniu do mechanizmu adresowania?

8.7. Czy jest konieczne, aby podczas wykonywania procesu wszystkie jego strony znajdowały się w pamięci głównej?

8.8. Czy strony procesu znajdujące się w pamięci głównej muszą sąsiadować ze sobą?

8.9. Czy jest konieczne, aby strony procesu znajdujące się w pamięci głównej były ułożone sekwencyjnie?

8.10. Jaki jest cel wprowadzenia bufora translacji adresów stron pamięci?

Problemy do rozwiązania

8.1. Załóżmy, że dysponujemy wieloprogramowym komputerem, w którym każde zadanie ma identyczne cechy. W jednym okresie obliczeń T połowa czasu jest zużywana przez wejście-wyjście, a połowa przez procesor. Każde zadanie jest aktywne łącznie przez N okresów. Załóżmy, że jest stosowany prosty, cykliczny schemat priorytetu, i że operacje we-wy mogą się nakładać z operacjami procesora. Zdefiniujmy następujące wielkości:

– długość cyklu przetwarzania = rzeczywisty czas zakończenia zadania;

– przepustowość = średnia liczba zadań kończonych w okresie T;

– wykorzystanie procesora = procent czasu, w którym procesor jest aktywny (nie czeka).

Oblicz te wielkości dla jednego, dwóch i czterech jednoczesnych zadań, zakładając, że okres T jest rozłożony następująco:

(a) wejście-wyjście pierwsza połowa, procesor druga;

(b) wejście-wyjście pierwsza i czwarta ćwiartka, procesor druga i trzecia.

8.2. Program zorientowany na wejście-wyjście jest to program, który (jeśli byłby realizowany sam) spędziłby więcej czasu, czekając na wejście-wyjście, niż używając procesora. Program zorientowany na procesor jest przeciwieństwem poprzedniego. Załóżmy, że algorytm szeregowania krótkookresowego faworyzuje te programy, które ostatnio zużywały niewiele czasu procesora. Wyjaśnij, dlaczego ten algorytm faworyzuje programy zorientowane na wejście-wyjście, a mimo to nie blokuje permanentnie programów zorientowanych na procesor.

8.3. Program oblicza sumy szeregu

$$C_i = \sum_{j=1}^{n} a_{ij}$$

dla tablicy A o wymiarze 100 na 100. Załóżmy, że komputer stosuje stronicowanie na żądanie z rozmiarem strony 1000 słów i że pojemność pamięci głównej związana z danymi wynosi 5 ramek. Czy wystąpiłaby różnica częstości błędów stron, jeśli A byłaby przechowywana w pamięci wirtualnej w postaci wierszy i kolumn? Wyjaśnij.

8.4. Załóżmy, że tablica stron procesu realizowanego na bieżąco przez procesor wygląda następująco. Wszystkie liczby są dziesiętne, wszystko jest numerowane począwszy od 0 i wszystkie adresy są bajtowymi adresami w pamięci. Rozmiar strony to 1024 bity.

Wirtualny numer strony	Bit ważności	Bit odniesienia	Bit modyfikacji	Numer ramki
0	1	1	0	4
1	1	1	1	7
2	0	0	0	–
3	1	0	0	2
4	0	0	0	–
5	1	0	1	0

(a) Opisz dokładnie, w jaki sposób – ogólnie rzecz biorąc – adres wirtualny generowany przez procesor jest tłumaczony na adres fizyczny w pamięci głównej. Sporządź listę wszystkich wirtualnych adresów, które mogą spowodować błędy stron.

(b) Jakie są adresy fizyczne w pamięci głównej (jeśli jakiekolwiek są) dla następujących adresów wirtualnych? (Nie próbuj przetwarzać żadnych błędów stron, jeśli takie istnieją).

(i) 1052
(ii) 2221
(iii) 5499

8.5. Podaj przyczyny, dla których rozmiar strony w systemie pamięci wirtualnej nie powinien być ani bardzo mały, ani bardzo duży.

8.6. Następująca sekwencja wirtualnych numerów stron wystąpiła podczas pracy komputera z pamięcią wirtualną:

3 4 2 6 4 7 1 3 2 6 3 5 1 2 3

Załóżmy, że jest stosowany algorytm wymiany stron, do których dawno nie było odwołania. Sporządź wykres współczynnika trafień stron (udziału odniesień do stron, przy których strona została znaleziona w pamięci głównej) w funkcji pojemności pamięci głównej mierzonej w stronach n dla $1 \leqslant n \leqslant 8$. Załóż, że pamięć główna jest początkowo pusta.

8.7. W systemie VAX tablice stron użytkownika są lokowane pod adresami wirtualnymi w przestrzeni systemowej. Na czym polega zaleta umieszczania tablic stron użytkownika w pamięci wirtualnej zamiast w głównej?

8.8. Rozważmy system komputerowy, w którym są stosowane segmentowanie i stronicowanie. Gdy segment znajduje się w pamięci, niektóre słowa na ostatniej stronie są tracone. Ponadto, przy rozmiarze segmentu s i strony p występuje s/p zapisów tablicy stron. Im mniejszy jest rozmiar strony, tym mniejsza jest strata na ostatniej stronie segmentu, jednak większa jest tablica stron. Jaki rozmiar strony minimalizuje łączne straty?

8.9. Komputer ma pamięć podręczną, pamięć główną i dyskową wykorzystywaną do pamięci wirtualnej. Jeśli słowo znajduje się w pamięci podręcznej, to jest wymaganych 20 ns w celu uzyskania dostępu do niego. Jeśli jest ono w pamięci głównej, ale nie w podręcznej, to najpierw potrzeba 60 ns na jego załadowania do pamięci podręcznej, po czym znów rozpoczyna się operacja odniesienia. Gdy słowo nie znajduje się w pamięci głównej, jest wymaganych 12 ms na pobranie go z dysku, po czym potrzeba 60 ns na umieszczenie go w pamięci podręcznej. Współczynnik trafień pamięci podręcznej wynosi 0,9, a współczynnik trafień pamięci głównej wynosi 0,6. Jaki jest średni czas dostępu do słowa w tym systemie?

8.10. Załóżmy, że zadanie jest dzielone na 4 równe segmenty i że system tworzy 8-zapisową tablicę deskryptorów stron dla każdego segmentu. System stanowi więc kombinację segmentowania i stronicowania. Załóżmy także, że rozmiar strony wynosi 2 KB.

(a) Jaki jest maksymalny rozmiar każdego segmentu?
(b) Jaka jest maksymalna przestrzeń adresów logicznych dla zadania?
(c) Załóżmy, że w zadaniu tym następuje dostęp do elementu zawartego w komórce fizycznej 00021 ABC. Jaki jest format adresu logicznego, który jest w tym celu generowany przez zadanie? Jaka jest maksymalna przestrzeń adresu fizycznego tego systemu?

8.11. Załóżmy, że pewien mikroprocesor może uzyskiwać dostęp do 2^{32} bajtów fizycznej pamięci głównej. Wykorzystuje on jedną, segmentowaną przestrzeń adresów logicznych o maksymalnym rozmiarze 2^{31} bajtów. Każdy rozkaz zawiera pełny, dwuczęściowy adres. Stosowane są zewnętrzne jednostki zarządzania pamięcią (MMU), których schemat zarządzania przypisuje segmentom kolejne bloki pamięci fizycznej o ustalonym rozmiarze 2^{22} bajtów. Początkowy adres fizyczny segmentu jest zawsze podzielny przez 1024. Pokaż szczegółowe powiązania zewnętrznego mechanizmu odwzorowania, który dokonuje konwersji adresów logicznych na fizyczne, stosując odpowiednią liczbę jednostek MMU. Pokaż szczegółową strukturę wewnętrzną MMU (zakładając, że każda MMU zawiera 128-zapisową pamięć podręczną bezpośrednio odwzorowanych deskryptorów segmentów) oraz jak jest wybierana każda MMU.

8.12. Rozważmy stronicowaną przestrzeń adresów logicznych (złożoną z 32 stron po 2 KB każda) odwzorowaną na przestrzeń 1-megabajtowej pamięci fizycznej.

(a) Jaki jest format adresu logicznego procesora?
(b) Jaka jest długość i szerokość tablicy stron (zaniedbując bity „praw dostępu")?
(c) Jaki wpływ na tablicę stron miałoby zmniejszenie przestrzeni pamięci fizycznej o połowę?

8.10. Załóżmy, że zadanie jest dzielone na 4 równe segmenty i że system tworzy 8-zapisową tablicę deskryptorów stron dla każdego segmentu. System stanowi więc kombinację segmentowania i stronicowania. Załóżmy także, że rozmiar strony wynosi 2 KB.

(a) Jaki jest maksymalny rozmiar każdego segmentu?

(b) Jaka jest maksymalna przestrzeń adresów logicznych dla zadania?

(c) Załóżmy, że w zadaniu tym następuje dostęp do elementu zawartego w komórce fizycznej 00021ABC. Jaki jest format adresu logicznego, który jest w tym celu generowany przez zadanie? Jaka jest maksymalna przestrzeń adresu fizycznego tego systemu?

8.11. Załóżmy, że pewien mikroprocesor może uzyskiwać dostęp do 2^{32} bajtów fizycznej pamięci głównej. Wykorzystuje on jedną, segmentowaną przestrzeń adresów logicznych o maksymalnym rozmiarze 2^{31} bajtów. Każdy rozkaz zawiera pełny, dwuczęściowy adres. Stosowane są zewnętrzne jednostki zarządzania pamięcią (MMU), których schemat zarządzania przypisuje segmentom kolejne bloki pamięci fizycznej o ustalonym rozmiarze 2^{22} bajtów. Początkowy adres fizyczny segmentu jest zawsze podzielny przez 1024. Pokaż szczegółowe powiązania zewnętrznego mechanizmu odwzorowania, który dokonuje konwersji adresów logicznych na fizyczne, stosując odpowiednią liczbę jednostek MMU. Pokaż szczegółową strukturę wewnętrzną MMU (zakładając, że każda MMU zawiera 128-zapisową pamięć podręczną bezpośrednio odwzorowanych deskryptorów segmentów) oraz jak jest wybierana każda MMU.

8.12. Rozważmy stronicowaną przestrzeń adresów logicznych (złożoną z 32 stron po 2 KB każda) odwzorowaną na przestrzeń 1-megabajtowej pamięci fizycznej.

(a) Jaki jest format adresu logicznego procesora?

(b) Jaka jest długość i szerokość tablicy stron (pomijając bity „praw dostępu")?

(c) Jaki wpływ na tablicę stron miałoby zmniejszenie przestrzeni pamięci fizycznej o połowę?

Część trzecia
Jednostka centralna

Problemy przedstawione w części trzeciej

Aż do tego miejsca rozpatrywaliśmy procesor w zasadzie jako „czarną skrzynkę" i rozważaliśmy jego oddziaływanie z wejściem-wyjściem oraz z pamięcią. W części trzeciej zajmiemy się strukturą wewnętrzną i działaniem procesora. Składa się on z jednostki sterującej, jednostki arytmetyczno-logicznej, jednostki wykonywania rozkazów i wzajemnych połączeń między tymi składnikami. Zostaną przedstawione zagadnienia architektury, takie jak projektowanie listy rozkazów i rodzaje danych. Będą również omówione problemy organizacyjne, takie jak przetwarzanie potokowe.

Tematyka poszczególnych rozdziałów części trzeciej

Rozdział 9. Arytmetyka komputera

W rozdziale 9 zostanie przeanalizowane działanie jednostki arytmetyczno-logicznej, przy czym skupimy się na reprezentowaniu liczb i na metodach implementacji operacji arytmetycznych. Procesory obsługują zwykle dwa rodzaje operacji arytmetycznych: całkowitoliczbowe i zmiennopozycyjne. W odniesieniu do obydwu tych rodzajów najpierw przeanalizujemy reprezentację liczb, następnie zaś operacje arytmetyczne. Szczegółowo zostanie przedstawiony ważny standard zmiennopozycyjny: IEEE 754.

Rozdział 10. Listy rozkazów: właściwości i funkcje

Z punktu widzenia programisty najlepszym sposobem zrozumienia działania procesora jest poznanie zbioru rozkazów maszynowych, jakie on wykonuje. Złożone zagadnienie projektowania list rozkazów zajmie rozdziały 10 i 11. W rozdziale 10 skupimy się na funkcjonalnych aspektach projektowania listy rozkazów. Przeanalizujemy rodzaje funkcji określanych za pomocą rozkazów komputerowych, po czym zajmiemy się w szczególności rodzajami argumentów (danymi, na których mają być wykonywane operacje) oraz rodzajami operacji (które mają być wykonywane), jakie zwykle znajdują się w listach rozkazów. Następnie krótko przeanalizujemy zależność między rozkazami procesora a językiem asemblerowym.

Rozdział 11. Listy rozkazów: tryby i formaty adresowania

Podczas gdy w rozdziale 10 zajęliśmy się raczej semantyką list rozkazów, w rozdziale 11 skupimy się na ich składni. W szczególności zajmiemy się sposobem określania adresów pamięci i ogólnym formatem rozkazów komputerowych.

Rozdział 12. Struktura i działanie jednostki centralnej

Rozdział 12 zostanie poświęcony omówieniu wewnętrznej struktury i działania procesora. Opiszemy w nim zastosowanie rejestrów jako wewnętrznej pamięci procesora, po czym będzie podsumowany dotychczas przedstawiony materiał, przez dokonanie przeglądu struktury i działania procesora. Pokażemy ogólną organizację procesora (jednostkę arytmetyczno-logiczną, jednostkę sterującą, plik rejestrów). Następnie zostanie przeanalizowana organizacja pliku rejestrów. W pozostałej części rozdziału będzie opisane działanie procesora podczas wykonywania rozkazów maszynowych. Przeanalizujemy cykl rozkazu w celu zademonstrowania funkcji i wzajemnych zależności między cyklami pobierania, pośredniego, wykonywania i przerwania. Na zakończenie dogłębnie przedstawimy przetwarzanie potokowe jako sposób zwiększania wydajności.

Rozdział 13. Komputery o zredukowanej liście rozkazów

W pozostałych rozdziałach części trzeciej zostaną bardziej szczegółowo przedstawione podstawowe tendencje rozwoju projektowania jednostek centralnych. W rozdziale 13 zostanie omówione podejście związane z koncepcją komputera o zredukowanej liście rozkazów (RISC), która stanowi najbardziej znaczącą innowację w dziedzinie organizacji i architektury komputerów, z jaką mieliśmy do czynienia w ostatnich latach. Architektura RISC stanowi radykalne odstępstwo od historycznej tendencji rozwoju architektury procesorów. Analiza tego podejścia skłania do rozważenia wielu ważnych zagadnień organizacji i architektury komputerów. W rozdziale tym przedstawimy uzasadnienie stosowania rozwiązania RISC, po czym szczegółowo omówimy projektowanie list rozkazów i architektury RISC, porównując to rozwiązanie z podejściem opartym na złożonej liście rozkazów (CISC).

Rozdział 14. Paralelizm na poziomie rozkazów i procesory superskalarne

W rozdziale 14 przeanalizujemy jeszcze nowsze i równie ważne innowacje projektowe: procesory superskalarne. Chociaż technologia superskalarna może być zastosowana w dowolnym procesorze, jest ona szczególnie dostosowana do architektury RISC. W rozdziale tym zostanie również przedstawione ogólne zagadnienie paralelizmu na poziomie rozkazów.

Rozdział 15. Architektura IA-64

Architektura listy rozkazów IA-64 stanowi nowe podejście do zapewniania wsparcia sprzętowego dla paralelizmu na poziomie rozkazów i znacznie różni się od architektury superskalarnej. Rozdział 15 rozpocznie się od omówienia czynników uzasadniających tę nową architekturę. Następnie pokażemy ogólną organizację, która ją wspiera. Przeanalizujemy również dość szczegółowo podstawowe cechy architektury IA-64 sprzyjające paralelizmowi na poziomie rozkazów.

Rozdział 9

Arytmetyka komputera

9.1. Jednostka arytmetyczno-logiczna

9.2. Reprezentacja liczb całkowitych

Reprezentacja znak-moduł
Reprezentacja uzupełnienia do dwóch
Konwersja między różnymi długościami bitowymi
Reprezentacja stałopozycyjna

9.3. Arytmetyka liczb całkowitych

Negowanie
Dodawanie i odejmowanie
Mnożenie
Dzielenie

9.4. Reprezentacja zmiennopozycyjna

Zasady
Norma IEEE dotycząca binarnej reprezentacji zmiennopozycyjnej

9.5. Arytmetyka zmiennopozycyjna

Dodawanie i odejmowanie
Mnożenie i dzielenie
Analiza dokładności
Norma IEEE dotycząca binarnej arytmetyki zmiennopozycyjnej

9.6. Polecana literatura i witryny WWW

9.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

PODSTAWOWE SPOSTRZEŻENIA

- Dwoma podstawowymi problemami arytmetyki komputera są: sposób reprezentowania liczb (w formacie binarnym) oraz algorytmy podstawowych operacji arytmetycznych (dodawania, odejmowania, mnożenia, dzielenia). Obydwa te problemy dotyczą zarówno arytmetyki całkowitoliczbowej, jak i zmiennopozycyjnej.
- Liczby zmiennopozycyjne mogą być wyrażane jako liczba (mantysa) pomnożona przez stałą (podstawę) podniesioną do pewnej potęgi (wykładnik). Liczby zmiennopozycyjne mogą służyć do wyrażania liczb bardzo wielkich i bardzo małych.
- W większości procesorów została implementowana norma IEEE 754 odnosząca się do reprezentacji i arytmetyki zmiennopozycyjnej. Norma ta określa zarówno format 32-bitowy, jak i 64-bitowy.

Rozpoczniemy analizowanie procesora od jednostki arytmetyczno-logicznej (ALU). Po krótkim wprowadzeniu skupimy się na najbardziej złożonym aspekcie ALU, a mianowicie na arytmetyce komputera. Funkcje logiczne, będące częścią ALU, są opisane w rozdz. 10, a wdrożenie prostych funkcji logicznych i arytmetycznych w postaci cyfrowych układów logicznych – w dodatku A do książki.

Arytmetyka komputera jest zwykle realizowana za pomocą dwóch bardzo różnych rodzajów liczb: całkowitych i zmiennopozycyjnych. W obu przypadkach wybór reprezentacji jest kluczowym problemem projektowym i zajmiemy się nim na początku, po czym omówimy operacje arytmetyczne.

W rozdziale tym podajemy przykłady, z których każdy jest wyróżniony za pomocą cieniowanej ramki.

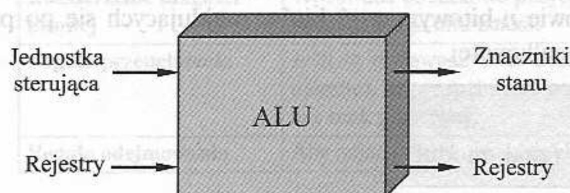
9.1. Jednostka arytmetyczno-logiczna

Jednostka arytmetyczno-logiczna (ALU) jest tą częścią komputera, która realizuje operacje arytmetyczne i logiczne na danych. Wszystkie inne elementy systemu komputerowego – jednostka sterująca, rejestry, pamięć, wejście-wyjście – istnieją głównie po to, żeby dostarczać dane do ALU w celu ich przetworzenia, a następnie odbierać wyniki. Rozpatrując ALU, sięgamy w pewnym sensie do rdzenia komputera.

Jednostka arytmetyczno-logiczna, a w rzeczywistości wszystkie podzespoły elektroniczne komputera, wykorzystują proste, cyfrowe układy logiczne, które mogą przechowywać cyfry binarne i wykonywać proste operacje logiczne Boole'a. W dodatku A do tej książki znajduje się opis realizacji cyfrowych układów logicznych.

Na rysunku 9.1 jest pokazany – w bardzo ogólnym ujęciu – sposób połączenia ALU z resztą procesora. Dane są przedstawiane ALU w rejestrach i również w rejestrach są przechowywane wyniki operacji. Rejestry te są miejscami tymczasowego przechowywania danych wewnątrz procesora i są połączone ścieżkami sygnałowymi

z ALU (zobacz np. rys. 2.3). ALU ustawia również znaczniki (*flags*) będące wynikiem operacji. Na przykład znacznik przepełnienia jest ustawiany jako 1, jeżeli wynik obliczeń przekracza długość rejestru, w którym ma być przechowywany. Wartości znaczników są również przechowywane w rejestrach wewnątrz procesora. Jednostka sterująca dostarcza sygnały, które sterują działaniem ALU oraz ruchem danych do (i z) ALU.



Rysunek 9.1. Wejścia i wyjścia ALU

9.2. Reprezentacja liczb całkowitych

W systemie binarnym[1] dowolne liczby mogą być reprezentowane po prostu za pomocą cyfr 0 i 1, znaku minus oraz przecinka. Na przykład

$$-1101,0101_2 = -11{,}3125_{10}$$

Dla celów przechowywania i przetwarzania w komputerze nie dysponujemy jednakże znakami minus i przecinka. Tylko cyfry binarne (0 i 1) mogą służyć do reprezentowania liczb. Gdybyśmy używali wyłącznie nieujemnych liczb całkowitych, reprezentacja byłaby prosta.

Słowo 8-bitowe mogłoby reprezentować liczby od 0 do 255, w tym:

00000000 = 0
00000001 = 1
00101001 = 41
10000000 = 128
11111111 = 255

Ogólnie, jeśli n-bitowy ciąg cyfr binarnych $a_{n-1}a_{n-2} \dots a_1a_0$ jest interpretowany jako pozbawiona znaku liczba całkowita A, to jej wartość oblicza się ze wzoru

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

[1] Podstawowe informacje o systemach liczbowych (binarnym, dziesiątkowym, szesnastkowym) znajdują się w dodatku B.

Reprezentacja znak-moduł

Istnieje kilka alternatywnych konwencji, które można zastosować do reprezentowania zarówno ujemnych, jak i dodatnich liczb całkowitych; wszystkie z nich polegają na traktowaniu najbardziej znaczącego (położonego po lewej stronie) bitu słowa jako bitu znaku: jeśli ten bit jest równy 0, to liczba jest dodatnia, a jeśli 1 – ujemna.

Najprostszą formą reprezentacji, w której jest używany bit znaku, jest reprezentacja znak-moduł. W słowie n-bitowym $n-1$ bitów znajdujących się po prawej stronie określa moduł liczby całkowitej.

+18 = 00010010
−18 = 10010010 (znak-moduł)

Ogólny przypadek można wyrazić następująco:

$$\textbf{Znak-moduł} \qquad A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{jeśli} \quad a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{jeśli} \quad a_{n-1} = 1 \end{cases} \tag{9.1}$$

Reprezentacja (notacja) znak-moduł ma kilka wad. Jedna polega na tym, że dodawanie i odejmowanie wymagają rozważania zarówno znaków liczb, jak i ich modułów w celu przeprowadzenia wymaganej operacji. Stanie się to jasne po zapoznaniu się z treścią podrozdz. 9.3. Inną wadą jest występowanie dwóch reprezentacji liczby 0:

$+0_{10}$ = 00000000
-0_{10} = 10000000 (znak-moduł)

Jest to niewygodne, ponieważ nieco trudniej jest sprawdzać 0 (operacja często przeprowadzana przez komputery), niż gdyby występowała jedna reprezentacja.

Ze względu na te wady reprezentacja znak-moduł rzadko jest używana do implementacji całkowitoliczbowej części jednostki arytmetyczno-logicznej. Zamiast niej stosuje się najczęściej reprezentację uzupełnienia do dwóch.

Reprezentacja uzupełnienia do dwóch

Podobnie jak w notacji znak-moduł, w reprezentacji uzupełnienia do dwóch używa się najbardziej znaczącego bitu jako bitu znaku, co ułatwia sprawdzanie, czy liczba całkowita jest dodatnia, czy ujemna. Natomiast różni się ona od reprezentacji znak-moduł sposobem interpretowania pozostałych bitów. W tabeli 9.1 przedstawiono podstawowe własności reprezentacji uzupełnienia do dwóch oraz opartej na niej arytmetyki; zagadnienia te zostaną omówione w bieżącym i w następnym podrozdziale.

Tabela 9.1. Właściwości reprezentacji uzupełnienia do dwóch i opartej na niej arytmetyce

Zakres	-2^{n-1} do $2^{n-1}-1$
Liczba reprezentacji zera	jedna
Negacja	Weź uzupełnienie boolowskie każdego bitu odpowiadającego liczbie dodatniej, po czym dodaj 1 do wynikłego stąd wzoru bitowego traktowanego jako bezznakowa liczba całkowita.
Rozszerzenie długości bitowej	Wprowadź dodatkowe pozycje bitowe po lewej i wypełnij je wartością początkowego bitu znaku.
Reguła przepełnienia	Jeśli są dodawane dwie liczby o takim samym znaku (dodatnie lub ujemne), to przepełnienie następuje wtedy i tylko wtedy, kiedy wynik ma znak przeciwny.
Reguła odejmowania	Aby odjąć B od A, weź uzupełnienie do dwóch liczby B i dodaj je do A.

Przy omawianiu reprezentacji uzupełnienia do dwóch autorzy skupiają się najczęściej na regułach przedstawiania liczb ujemnych, bez jakiegokolwiek formalnego dowodu, że określony sposób „działa". W przeciwieństwie do tego, przedstawienie liczb całkowitych w formie uzupełnienia do dwóch w podrozdziale bieżącym i w podrozdz. 9.3 jest oparte na [DATT93]; w tym źródle sugeruje się, że najlepiej można zrozumieć tę reprezentację, definiując ją w kategoriach ważonej sumy bitów, podobnie jak uczyniliśmy to w odniesieniu do reprezentacji bezznakowej oraz znak-moduł. Zaletą takiego ujęcia jest brak jakichkolwiek uporczywych wątpliwości, że w jakichś szczególnych przypadkach operacje arytmetyczne w notacji uzupełnienia do dwóch mogą nie funkcjonować.

Rozważmy n-bitową liczbę całkowitą A wyrażoną za pomocą uzupełnienia do dwóch. Jeśli A jest dodatnia, to bit znaku a_{n-1} ma wartość 0. Pozostałe bity reprezentują moduł liczby w ten sam sposób jak w przypadku notacji znak-moduł

$$A = \sum_{i=0}^{n-2} 2^i a_i \qquad \text{dla}\, A \geqslant 0$$

Liczba zer jest określana jako dodatnia; wobec tego ma bit znaku równy 0 i moduł złożony z samych zer. Jak widać, zakres dodatnich liczb całkowitych, które mogą być reprezentowane, sięga od 0 (wszystkie bity modułu są 0) do $2^{n-1}-1$ (wszystkie bity modułu są 1). Jakakolwiek większa liczba wymagałaby więcej bitów.

W przypadku ujemnej liczby A ($A < 0$) bit znaku a_{n-1} jest równy 1. Pozostałe $n-1$ bitów może przybierać dowolną z 2^{n-1} wartości. Wobec tego zakres ujemnych liczb całkowitych, które mogą być reprezentowane, sięga od -1 do -2^{n-1}. Chcielibyśmy przypisać wartości bitowe ujemnym liczbom całkowitym w taki sposób, aby operacje arytmetyczne mogły być realizowane w sposób prosty, podobnie jak w przypadku bezznakowych liczb całkowitych. W tym ostatnim przypadku obliczenie wartości liczby całkowitej na podstawie jej reprezentacji bitowej wymaga wagi najbardziej znaczącego bitu równej $+2^{n-1}$. Jak się okaże w podrozdz. 9.3, osiągnięcie pożądanych własności arytmetycznych w odniesieniu do reprezentacji bitu znaku uzyskuje się wówczas, gdy wagą najbardziej znaczącego bitu jest -2^{n-1}. Taka właśnie jest kon-

wencja stosowana w reprezentacji uzupełnienia do dwóch, co prowadzi do następującego wyrażenia na liczby ujemne:

$$\textbf{Uzupełnienie do dwóch} \qquad A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i \tag{9.2}$$

W przypadku dodatnich liczb całkowitych $a_{n-1}=0$, więc $-2^{n-1}a_{n-1}=0$. Wobec tego równanie (9.1) definiuje reprezentację uzupełnienia do dwóch zarówno liczb dodatnich, jak i ujemnych.

Tabela 9.2. Alternatywne reprezentacje 4-bitowych liczb całkowitych

Reprezentacja dziesiętna	Reprezentacja znak-moduł	Reprezentacja uzupełnienia do dwóch	Reprezentacja przesunięta
+8	–	–	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
–0	1000	–	–
–1	1001	1111	0110
–2	1010	1110	0101
–3	1011	1101	0100
–4	1100	1100	0011
–5	1101	1011	0010
–6	1110	1010	0001
–7	1111	1001	0000
–8	–	1000	–

W tabeli 9.2 znajduje się porównanie reprezentacji znak-moduł oraz uzupełnienia do dwóch w odniesieniu do 4-bitowych liczb całkowitych. Chociaż uzupełnienie do dwóch stanowi kłopotliwą reprezentację z punktu widzenia człowieka, zobaczymy, że ułatwia ona implementację najważniejszych operacji arytmetycznych, dodawania i odejmowania. Z tego powodu jest ono niemal uniwersalnie używane do reprezentowania liczb całkowitych w procesorach.

Użyteczną ilustracją natury reprezentacji uzupełnienia do dwóch jest tablica wartości, w której wartość znajdująca się przy prawej krawędzi tablicy wynosi 1 (2^0), a każda kolejna wartość położona na lewo jest podwojona, aż do wartości położonej

najbardziej na lewo, która jest zanegowana. Jak można zauważyć na rys. 9.2a, największa ujemna liczba, która może być reprezentowana w notacji uzupełnienia do dwóch, jest równa -2^{n-1}. Jeśli którykolwiek z bitów poza bitem znaku jest równy 1, dodaje on do liczby pewną dodatnią wartość. Jest również jasne, że liczba ujemna musi mieć 1 po lewej stronie, a liczba dodatnia musi mieć na tej pozycji 0. Największa liczba dodatnia jest więc reprezentowana przez ciąg zaczynających się od 0, po którym następują same 1; jest ona równa $2^{n-1} - 1$.

–128	64	32	16	8	4	2	1

(a) ośmiopozycyjna tablica wartości uzupełnienia do dwóch

–128	64	32	16	8	4	2	1	
1	0	0	0	0	0	1	1	
–128						+2	+1	= –125

(b) konwersja liczby binarnej 10000011 na dziesiętną

	–128	64	32	16	8	4	2	1
	1	0	0	0	1	0	0	0
–120 =	–128				+8			

(c) konwersja liczby dziesiętnej –120 na binarną

Rysunek 9.2. Zastosowanie tablicy wartości do konwersji między reprezentacją uzupełnienia do dwóch a reprezentacją dziesiętną

W pozostałej części rys. 9.2 zostało zilustrowane zastosowanie tablicy wartości do konwersji liczb między notacją uzupełnienia do dwóch a dziesiętną.

Konwersja między różnymi długościami bitowymi

Jest czasem pożądane przechowanie całkowitej liczby n-bitowej w postaci m bitów, gdzie $m > n$. W przypadku notacji znak-moduł można tego dokonać z łatwością: po prostu trzeba przesunąć bit znaku do najdalszej lewej pozycji oraz wypełnić pozostałe wolne pozycje zerami.

+18	=	00010010	(znak-moduł, 8 bitów)
+18	=	0000000000010010	(znak-moduł, 16 bitów)
–18	=	10010010	(znak-moduł, 8 bitów)
–18	=	1000000000010010	(znak-moduł, 16 bitów)

Procedura taka nie działa w przypadku całkowitych liczb ujemnych w reprezentacji uzupełnienia do dwóch. Postępując jak w powyższym przykładzie, otrzymamy:

+18	=	00010010	(uzupełnienie do dwóch, 8 bitów)
+18	=	0000000000010010	(uzupełnienie do dwóch, 16 bitów)
–18	=	11101110	(uzupełnienie do dwóch, 8 bitów)
–32 658	=	1000000001101110	(uzupełnienie do dwóch, 16 bitów)

Przedostatni wiersz można z łatwością zidentyfikować za pomocą tablicy wartości z rys. 9.2. Natomiast ostatni może być zweryfikowany za pomocą równania 9.2 lub 16-bitowej tablicy wartości.

Zamiast tego, w przypadku uzupełnienia do dwóch stosuje się następującą regułę: wykonuje się przesunięcie bitu znaku do najdalszej lewej pozycji, po czym wypełnia się powstałe puste pozycje kopiami bitu znaku. W przypadku liczb dodatnich wypełnia się zerami, w przypadku ujemnych – jedynkami. Mamy więc:

–18	=	11101110	(uzupełnienie do dwóch, 8 bitów)
–18	=	1111111111101110	(uzupełnienie do dwóch, 16 bitów)

Żeby zobaczyć, dlaczego ta reguła funkcjonuje, ponownie rozważmy n-bitowy ciąg cyfr binarnych $a_{n-1} a_{n-2} \ldots a_1 a_0$ interpretowany jako liczba całkowita A w notacji uzupełnienia do dwóch. Jej wartość wynosi

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Jeśli A jest liczbą dodatnią, reguła oczywiście się sprawdza. Załóżmy teraz, że A jest ujemna i że chcemy zbudować m-bitową reprezentację przy $m > n$. Wtedy

$$A = -2^{m-1} a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i$$

Te dwie wartości muszą być równe:

$$-2^{m-1} + \sum_{i=0}^{m-2} 2^i a_i = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

$$-2^{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i = -2^{n-1}$$

$$2^{n-1} + \sum_{i=n-1}^{m-2} 2^i a_i = 2^{m-1}$$

$$1 + \sum_{i=0}^{n-2} 2^i + \sum_{i=n-1}^{m-2} 2^i a_i = 1 + \sum_{i=0}^{m-2} 2^i$$

$$\sum_{i=n-1}^{m-2} 2^i a_i = \sum_{i=n-1}^{m-2} 2^i$$

$$\Rightarrow a_{m-2} = \ldots = a_{n-2} = a_{n-1} = 1$$

Przechodząc od pierwszego do drugiego równania, wymagamy, żeby $n-1$ najmniej znaczących bitów nie zmieniało się przy przejściu od jednej reprezentacji do drugiej. Następnie przechodzimy do ostatniego równania, które jest prawdziwe tylko wtedy, gdy wszystkie bity na pozycjach od $n-1$ do $m-2$ są równe 1. Więc reguła poszerzania znaku działa.

Reprezentacja stałopozycyjna

Zwróćmy uwagę na to, że reprezentacje analizowane w tym podrozdziale są czasem określane jako stałopozycyjne. Jest tak, ponieważ położenie przecinka pozycyjnego (przecinka binarnego) jest ustalone i zakładamy, że znajduje się on na prawo od cyfry położonej najdalej na prawo. Programista może używać tej samej reprezentacji dla ułamków binarnych, posługując się skalowaniem liczb, tak że przecinek binarny przez implikację znajduje się w pewnym innym położeniu.

9.3. Arytmetyka liczb całkowitych

W tym podrozdziale zajmiemy się zwykłymi funkcjami arytmetycznymi na liczbach w reprezentacji uzupełnienia do dwóch.

Negowanie

W przypadku reprezentacji znak-moduł reguła negowania liczby całkowitej jest prosta: odwrócić bit znaku. W notacji uzupełnienia do dwóch negowanie liczby całkowitej może być dokonywane za pomocą następujących reguł:

1. Weź uzupełnienie Boole'a każdego bitu liczby całkowitej (łącznie z bitem znaku). Oznacza to ustawienie każdej 1 jako 0 oraz każdego 0 jako 1.
2. Traktując wynik jako liczbę całkowitą pozbawioną znaku, dodaj 1.

Taki dwuetapowy proces jest określany jako **operacja uzupełniania do dwóch** lub obliczanie uzupełnienia do dwóch liczby całkowitej.

+18	= 00010010	(uzupełnienie do dwóch)
uzupełnienie bitowe	= 11101101	
	+ 1	
	11101110 = −18	

Jak można było się spodziewać, negacją negacji liczby jest ona sama:

−18 =	11101110	(uzupełnienie do dwóch)
uzupełnienie bitowe =	00010001	
	+ 1	
	00010010 = 18	

Możemy zademonstrować słuszność opisanej powyżej metody, używając definicji reprezentacji uzupełnienia do dwóch w postaci równania (9.2). Ponownie zinterpretujmy n-bitowy ciąg cyfr binarnych $a_{n-1}\, a_{n-2} \ldots\, a_1 a_0$ jako liczbę całkowitą A w reprezentacji uzupełnienia do dwóch. Jej wartość wynosi więc

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Utwórzmy teraz uzupełnienie na poziomie bitowym $\overline{a_{n-1}}\, \overline{a_{n-2}} \ldots \overline{a_0}$ i traktując je jako pozbawioną znaku liczbę całkowitą, dodajmy 1. Na zakończenie zinterpretujmy powstały ciąg cyfr binarnych jako liczbę całkowitą B przedstawioną w notacji uzupełnienia do dwóch. Jej wartość jest następująca:

$$B = -2^{n-1} \overline{a_{n-1}} + 1 + \sum_{i=0}^{n-2} 2^i \overline{a_i}$$

Chcemy, żeby $A = -B$, co oznacza, że $A + B = 0$. Z łatwością możemy to wykazać:

$$A + B = -(a_{n-1} + \overline{a_{n-1}})2^{n-1} + 1 + \left(\sum_{i=0}^{n-2} 2^i (a_i + \overline{a_i})\right) = -2^{n-1} + 1 + \left(\sum_{i=0}^{n-2} 2^i\right) = -2^{n-1} + 2^{n-1} = 0$$

W powyższych obliczeniach założyliśmy, że możemy najpierw potraktować bitowe uzupełnienie A jako pozbawioną znaku liczbę całkowitą, do której dodajemy 1, po czym przyjmujemy, że wynik jest liczbą całkowitą w reprezentacji uzupełnienia do dwóch. Do rozważenia pozostają dwa przypadki szczególne. Najpierw weźmy $A = 0$. W tym przypadku przy reprezentacji 8-bitowej

0 =	00000000	(uzupełnienie do dwóch)
uzupełnienie bitowe =	11111111	
	+ 1	
	100000000 = 0	

Występuje tu *przepełnienie* (nadmiar), które jest ignorowane. Wynikiem negacji zera jest zero, tak jak powinno być.

Drugi przypadek szczególny stanowi większy problem. Jeśli zanegujemy wzór bitowy ciągu składającego się z 1 z następującymi $n - 1$ zerami, to otrzymamy tę samą liczbę. Na przykład:

```
                 -128 = 10000000    (uzupełnienie do dwóch)
uzupełnienie bitowe = 01111111
                    +        1
                      10000000 = -128
```

Pewna anomalia tego rodzaju jest nie do uniknięcia. Liczba możliwych wzorów bitowych w słowie n-bitowym wynosi 2^n i jest to liczba parzysta. Chcemy reprezentować dodatnie i ujemne liczby całkowite oraz zero. Jeśli jednakowe ilości liczb dodatnich i ujemnych są reprezentowane przez znak i moduł, to występują dwie reprezentacje zera. Jeśli natomiast jest tylko jedna reprezentacja zera (uzupełnienie do dwóch), to musi występować nierówna liczba reprezentowanych liczb dodatnich i ujemnych. W przypadku uzupełniania do dwóch istnieje n-bitowa reprezentacja dla -2^n, ale nie dla 2^n.

Dodawanie i odejmowanie

Dodawanie w notacji uzupełniania do dwóch jest pokazane na rys. 9.3. Pierwsze cztery przykłady ilustrują operacje udane. Jeśli wynik operacji jest dodatni, to otrzymujemy dodatnią liczbę w zwykłej notacji binarnej. Jeśli wynik operacji jest ujemny, to otrzymujemy liczbę ujemną w reprezentacji uzupełnienia do dwóch. Zauważmy, że w pewnych przypadkach poza końcem słowa występuje bit przeniesienia. Jest on ignorowany.

1001 = -7 +0101 = 5 1110 = -2 (a)(-7) + (+5)	1100 = -4 +0100 = 4 10000 = 0 (b) (-4) + (4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = -4 +1111 = -1 11011 = -5 (d) (-4) + (-1)
0101 = 5 +0100 = 4 1001 = Przepełnienie (e) (+5) + (+4)	1001 = -7 +1010 = -6 10011 = Przepełnienie (f) (-7) + (-6)

Rysunek 9.3. Dodawanie liczb w reprezentacji uzupełnienia do dwóch

Przy dodawaniu wynik może być większy, niż można zmieścić w ramach przyjętej długości słowa. Warunek ten jest nazywany **przepełnieniem**. Gdy występuje przepełnienie, ALU musi sygnalizować ten fakt, nie są więc podejmowane próby wykorzystania wyniku. W celu wykrywania przepełnienia stosowana jest następująca reguła: jeśli są dodawane dwie liczby i obie są albo dodatnie, albo ujem-

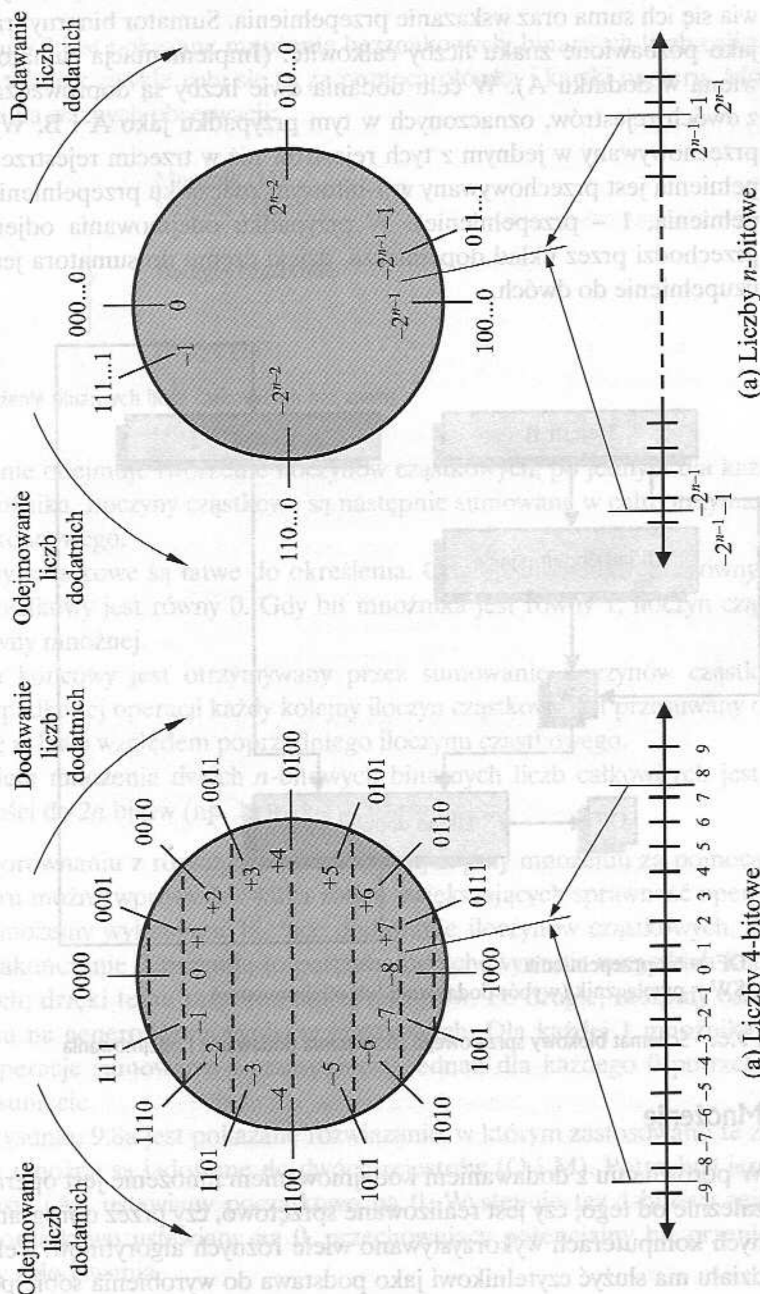
ne, to przepełnienie występuje wtedy i tylko wtedy, gdy wynik ma znak przeciwny. Przykłady przepełnienia zostały pokazane na rys. 9.3e i f. Zwróćmy uwagę, że przepełnienie może mieć miejsce zarówno wówczas, gdy występuje przeniesienie, jak i przy braku przeniesienia.

Odejmowanie jest również łatwe; wykonuje się je, używając następującej reguły: w celu odjęcia od jednej liczby (odjemnej) drugiej liczby (odjemnika) weź uzupełnienie do dwóch odjemnika i dodaj je do odjemnej. Odejmowanie jest więc realizowane poprzez dodawanie, co widać na rys. 9.4. Ostatnie dwa przykłady pokazują, że reguła przepełnienia nadal ma zastosowanie.

0010 = 2 +1001 = -7 1011 = -5 (a) M = 2 = 0010 S = 7 = 0111 -S = 1001	0101 = 5 +1110 = -2 10011 = 3 (b) M = 5 = 0101 S = 2 = 0010 -S = 1110
1011 = -5 +1110 = -2 11001 = -7 (c) M = -5 = 1011 S = 2 = 0010 -S = 1110	0101 = 5 +0010 = 2 0111 = 7 (d) M = 5 = 0101 S = -2 = 1110 -S = 0010
0111 = 7 +0111 = 7 1110 = Przepełnienie (e) M = 7 = 0111 S = -7 = 1001 -S = 0111	1010 = -6 +1110 = -4 10110 = Przepełnienie (f) M = -6 = 1010 S = 4 = 0100 -S = 1100

Rysunek 9.4. Odejmowanie liczb w reprezentacji uzupełnienia do dwóch (M-S)

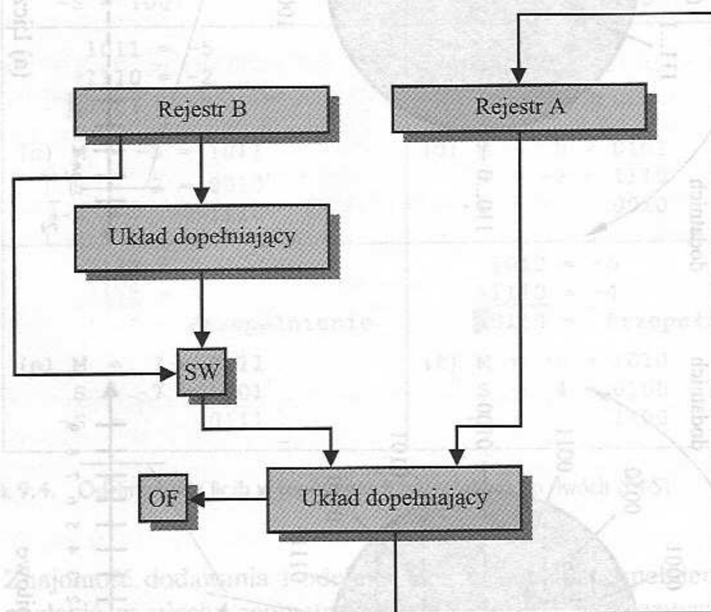
Znajomość dodawania i odejmowania w notacji uzupełnienia do dwóch można pogłębić w ujęciu geometrycznym [BENH92], pokazanym na rys. 9.5. Okręgi w górnej części każdego rysunku utworzono, wybierając odpowiednie segmenty ciągu liczb i łącząc ich końce. Zauważmy, że gdy liczby są umieszczone na okręgu, uzupełnienia do dwóch dowolnej liczby leżą po stronie przeciwnej w poziomie (co zostało pokazane za pomocą linii przerywanych). Poczynając od dowolnej liczby na okręgu, możemy dodać dodatnią liczbę k (lub odjąć ujemną liczbę k) do tej liczby, przenosząc się o k pozycji zgodnie z ruchem wskazówek zegara. Podobnie możemy odjąć dodatnią k (lub dodać ujemną k), przenosząc się o k pozycji przeciwnie do ruchu wskazówek zegara. Jeśli wynik operacji arytmetycznej prowadzi do przekroczenia punktu połączenia ciągu liczb na okręgu, odpowiedź jest błędna (przepełnienie).



Rysunek 9.5. Ilustracja geometryczna liczb w notacji uzupełnienia do dwóch

Wszystkie przykłady z rys. 9.3 i 9.4 można z łatwością prześledzić na rys. 9.5.

Na rysunku 9.6 są pokazane ścieżki danych i elementy sprzętowe wymagane do dodawania i odejmowania. Centralnym elementem jest sumator binarny, do którego są doprowadzane dwie liczby przeznaczone do dodania. Na jego wyjściu pojawia się ich suma oraz wskazanie przepełnienia. Sumator binarny traktuje obie liczby jako pozbawione znaku liczby całkowite. (Implementacja sumatora jest przedstawiona w dodatku A). W celu dodania dwie liczby są doprowadzane do sumatora z dwóch rejestrów, oznaczonych w tym przypadku jako A i B. Wynik częściej jest przechowywany w jednym z tych rejestrów niż w trzecim rejestrze. Wskaźnik przepełnienia jest przechowywany w 1-bitowym znaczniku przepełnienia (0 – brak przepełnienia, 1 – przepełnienie). W przypadku odejmowania odjemnik (rejestr B) przechodzi przez układ dopełniacza, dzięki czemu do sumatora jest doprowadzane uzupełnienie do dwóch.



Rysunek 9.6. Schemat blokowy sprzętowego rozwiązania dodawania i odejmowania

Mnożenie

W porównaniu z dodawaniem i odejmowaniem mnożenie jest operacją złożoną niezależnie od tego, czy jest realizowane sprzętowo, czy przez oprogramowanie. W różnych komputerach wykorzystywano wiele różnych algorytmów. Tekst tego podrozdziału ma służyć czytelnikowi jako podstawa do wyrobienia sobie poglądu na temat

zwykle stosowanego podejścia. Rozpoczniemy od prostego problemu mnożenia dwóch bezznakowych (nieujemnych) liczb całkowitych, po czym zapoznamy się z najbardziej powszechnymi metodami mnożenia liczb w reprezentacji uzupełnienia do dwóch.

Bezznakowe liczby całkowite

Na rysunku 9.7 jest pokazane mnożenie bezznakowych, binarnych liczb całkowitych wykonane tak, jak zwykle robi się to za pomocą ołówka i kartki papieru. Można tu poczynić kilka ważnych obserwacji:

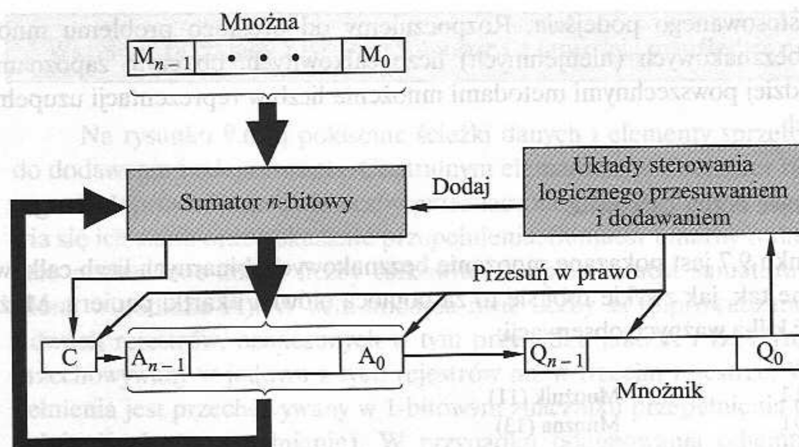
```
    1011      Mnożnik (11)
   ×1101      Mnożna (13)
    1011  ┐
   0000   │   Iloczyny cząstkowe
  1011    │
 1011     ┘
10001111      Iloczyn (143)
```

Rysunek 9.7. Mnożenie binarnych liczb całkowitych bez znaku

1. Mnożenie obejmuje tworzenie iloczynów cząstkowych, po jednym dla każdej cyfry mnożnika. Iloczyny cząstkowe są następnie sumowane w celu otrzymania iloczynu końcowego.
2. Iloczyny cząstkowe są łatwe do określenia. Gdy bit mnożnika jest równy 0, iloczyn cząstkowy jest równy 0. Gdy bit mnożnika jest równy 1, iloczyn cząstkowy jest równy mnożnej.
3. Iloczyn końcowy jest otrzymywany przez sumowanie iloczynów cząstkowych. W przypadku tej operacji każdy kolejny iloczyn cząstkowy jest przesuwany o jedną pozycję w lewo względem poprzedniego iloczynu cząstkowego.
4. Wynikiem mnożenia dwóch n-bitowych binarnych liczb całkowitych jest liczba o długości do $2n$ bitów (np. $11 \times 11 = 1001$).

W porównaniu z rozwiązaniem stosowanym przy mnożeniu za pomocą ołówka i papieru można wprowadzić kilka zmian zwiększających sprawność operacji. Po pierwsze, możemy wykonywać bieżące dodawanie iloczynów cząstkowych, nie czekając na zakończenie. Eliminuje to potrzebę przechowywania wszystkich iloczynów cząstkowych; dzięki temu potrzeba mniej rejestrów. Po drugie, możemy oszczędzić nieco czasu na generowaniu wyników cząstkowych. Dla każdej 1 mnożnika są wymagane operacje sumowania i przesunięcia; jednak dla każdego 0 potrzebne jest tylko przesunięcie.

Na rysunku 9.8a jest pokazane rozwiązanie, w którym zastosowano te zmiany. Mnożnik i mnożna są ładowane do dwóch rejestrów (Q i M). Potrzebny jest także trzeci rejestr (A), ustawiany początkowo na 0. Występuje też 1-bitowy rejestr C, również początkowo ustawiany na 0, przechowujący potencjalny bit przeniesienia wynikający z dodawania.



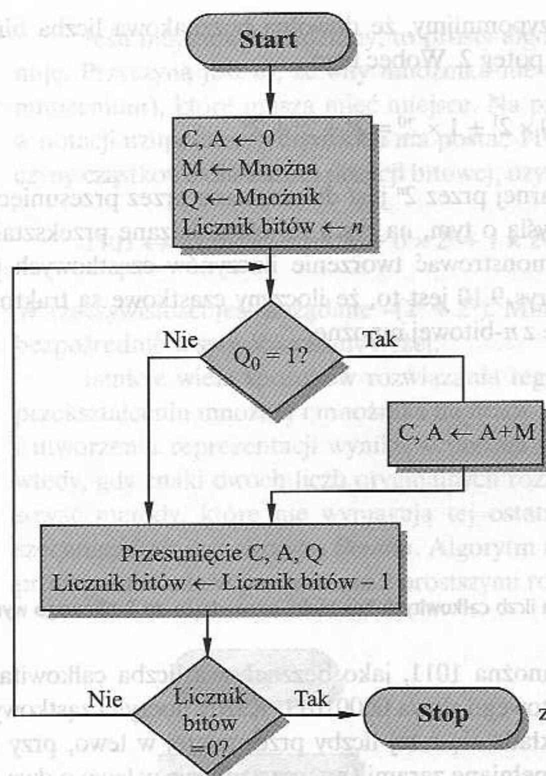
(a) Schemat blokowy

C	A	Q	M		
0	0000	1101	1011	Wartości początkowe	
0	1011	1101	1011	Dodawanie	Pierwszy cykl
0	0101	1110	1011	Przesuwanie	
0	0010	1111	1011	Przesuwanie	Drugi cykl
0	1001	1111	1011	Dodawanie	Trzeci cykl
0	0110	1111	1011	Przesuwanie	
1	0001	1111	1011	Dodawanie	Czwarty cykl
0	1000	1111	1011	Przesuwanie	

(b) Przykład z rys. 9.6 (iloczyn znajduje się w A, Q)

Rysunek 9.8. Sprzętowe wdrożenie mnożenia liczb binarnych bez znaku

Działanie układu mnożącego jest następujące. Układy logiczne sterowania odczytują mnożnik bit po bicie. Jeśli Q_0 jest równe 1, to mnożna jest dodawana do zawartości rejestru A i wynik pozostaje w rejestrze A. Następnie wszystkie bity rejestrów C, A i Q są przesuwane na prawo o jeden bit, przez co bit C przechodzi do A_{n-1}, A_0 przechodzi do Q_{n-1}, a bit Q_0 jest tracony. Jeśli Q_0 jest równy 0, to nie jest wykonywane żadne dodawanie, a jedynie przesunięcie. Proces ten jest powtarzany dla każdego bitu oryginalnego mnożnika. Wynikający stąd $2n$-bitowy iloczyn jest zawarty w rejestrach A i Q. Sieć działań dla tej operacji widać na rys. 9.9, przykład zaś na rys. 9.8b. Zauważmy, że w drugim cyklu, gdy bit mnożnika jest 0, nie ma operacji dodawania.



Rysunek 9.9. Sieć działań mnożenia liczb binarnych bez znaku

Mnożenie w notacji uzupełnienia do dwóch

Widzieliśmy, że dodawanie i odejmowanie liczb wyrażonych w notacji uzupełnienia do dwóch może być wykonywane tak, jak gdyby były one bezznakowymi liczbami całkowitymi. Rozważmy następujący przykład:

```
 1001
+0011
 1100
```

Jeśli te liczby są traktowane jako bezznakowe liczby całkowite, to dodajemy 9 (1001) do 3 (0011), otrzymując 12 (1100). Jeśli są to liczby całkowite w notacji uzupełnienia do dwóch, to dodajemy –7 (1001) do 3 (0011), otrzymując – 4 (1100).

Niestety, ten prosty schemat nie funkcjonuje w przypadku mnożenia. Żeby to stwierdzić, ponownie przeanalizujmy rys. 9.7 Pomnożyliśmy 11 (1011) przez 13 (1101), otrzymując 143 (10001111). Jeśli zinterpretujemy te liczby jako zapisane w notacji uzupełnienia do dwóch, to mamy –5 (1011) razy –3 (1101) równe –113 (10001111). Jak widać, ten prosty algorytm mnożenia nie funkcjonuje, jeśli mnożna i mnożnik są ujemne. W rzeczywistości nie działa to też poprawnie, gdy którekolwiek z nich jest ujemne. Aby to wyjaśnić, musimy wrócić do rys. 9.7 i zobaczyć, co dzieje się podczas

tej operacji z potęgami 2. Przypomnijmy, że dowolna bezznakowa liczba binarna może być wyrażona jako suma potęg 2. Wobec tego

$$1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 2^3 + 2^2 + 2^0$$

Ponadto, mnożenie liczby binarnej przez 2^n jest dokonywane przez przesunięcie tej liczby w lewo o n bitów. Z myślą o tym, na rys. 9.10 jest pokazane przekształcone działanie z rys. 9.7, aby zademonstrować tworzenie iloczynów cząstkowych przez mnożenie. Jedyną różnicą na rys. 9.10 jest to, że iloczyny cząstkowe są traktowane jako liczby $2n$-bitowe tworzone z n-bitowej mnożnej.

```
    1001
   ×1101
00001011     1011×1×2^0
00000000     1011×0×2^1
00101100     1011×1×2^2
01011000     1011×1×2^3
10001111
```

Rysunek 9.10. Mnożenie dwóch 4-bitowych liczb całkowitych bez znaku prowadzące do 8-bitowego wyniku

Wobec tego 4-bitowa mnożna 1011, jako bezznakowa liczba całkowita, jest przechowywana w postaci 8-bitowego słowa 00001011. Każdy iloczyn cząstkowy (inny niż odnoszący się do 2^0) składa się z tej liczby przesuniętej w lewo, przy czym wolne miejsca po prawej są wypełniane zerami (np. przesunięcie w lewo o dwa miejsca daje 00101100).

Możemy teraz zademonstrować, że prosty algorytm mnożenia nie będzie funkcjonował, jeśli mnożna jest ujemna. Problemem jest to, że każdy wkład ujemnej mnożnej w postaci iloczynu cząstkowego musi być liczbą ujemną na polu $2n$-bitowym; bity znaków iloczynów cząstkowych muszą pozostawać w szeregu. Można to zobaczyć na rys. 9.11, na którym jest pokazane mnożenie 1001 przez 0011. Jeśli liczby te są traktowane jako bezznakowe liczby całkowite, to mnożenie $9 \times 3 = 27$ przebiega prosto. Jeżeli jednak 1001 jest interpretowana jako –7 w notacji uzupełnienia do dwóch, to każdy iloczyn cząstkowy musi być ujemną liczbą $2n$-bitową w notacji uzupełnienia do dwóch, co widać na rys. 9.11b. Zauważmy, że można to uzyskać przez przesunięcie każdego iloczynu cząstkowego w lewo za pomocą binarnych 1.

(a) liczby całkowite bez znaku	(b) liczby całkowite w notacji uzupełnienia do dwóch
1001 (9)	1001 (-7)
×0011 (3)	×0011 (3)
00001001 1001 × 2^0	11111001 (-7) × 2^0 = (-7)
00010010 1001 × 2^1	11110010 (-7) × 2^1 = (-14)
00011011 (27)	11101011 (-21)

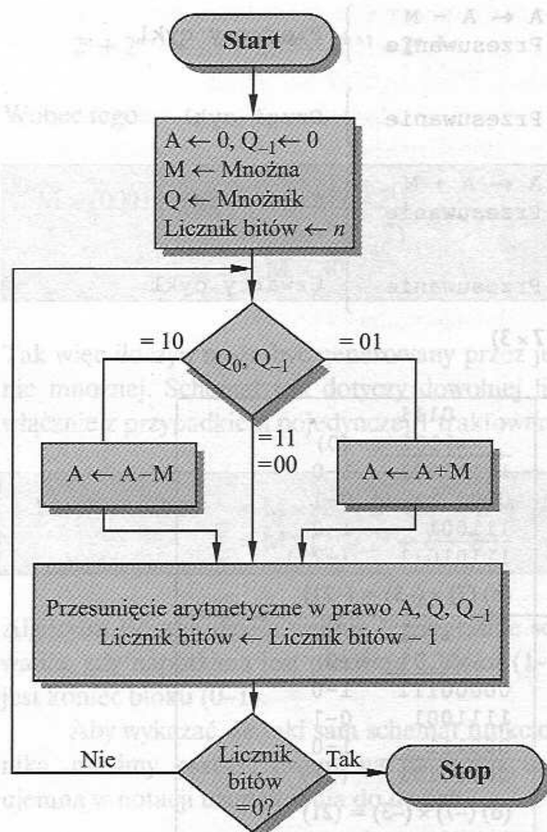
Rysunek 9.11. Porównanie mnożenia liczb całkowitych bez znaku i liczb całkowitych w notacji uzupełnienia do dwóch

Jeśli mnożnik jest ujemny, to prosty algorytm mnożenia również nie funkcjonuje. Przyczyną jest to, że bity mnożnika nie odpowiadają już przesunięciom (lub mnożeniom), które muszą mieć miejsce. Na przykład 4-bitowa liczba dziesiętna –3 w notacji uzupełnienia do dwóch ma postać 1101. Gdybyśmy po prostu obliczyli iloczyny cząstkowe dla każdej pozycji bitowej, uzyskalibyśmy:

$$1101 \leftrightarrow -(1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = -(2^3 + 2^2 + 2^0)$$

W rzeczywistości jest pożądane $-(2^1 + 2^0)$. Mnożnik ten nie może więc być używany bezpośrednio w sposób opisany wyżej.

Istnieje wiele sposobów rozwiązania tego problemu. Jeden z nich polega na przekształceniu mnożnej i mnożnika na liczby dodatnie, przeprowadzeniu mnożenia i utworzeniu reprezentacji wyniku w postaci uzupełnienia do dwóch wtedy i tylko wtedy, gdy znaki dwóch liczb oryginalnych różniły się. Projektanci wolą jednak stosować metody, które nie wymagają tej ostatniej transformacji. Jedną z najpowszechniejszych jest *algorytm Bootha*. Algorytm ten ma także tę zaletę, że przyspiesza proces mnożenia w porównaniu z prostszymi rozwiązaniami.



Rysunek 9.12. Algorytm Bootha mnożenia liczb w notacji uzupełnienia do dwóch

Algorytm Bootha jest przedstawiony na rys. 9.12. Można go opisać następująco. Jak poprzednio, mnożnik i mnożna są umieszczane odpowiednio w rejestrach Q i M. Istnieje także rejestr 1-bitowy, umieszczony logicznie na prawo od najmniej znaczącego bitu (Q_0) rejestru Q i oznaczony Q_{-1}; jego zastosowanie wyjaśnimy wkrótce. Wyniki mnożenia będą się pojawiały w rejestrach A i Q. Rejestry A i Q_{-1} są początkowo ustawiane na 0. Jak poprzednio, układy logiczne sterowania przeglądają mnożnik bit po bicie. Podczas badania każdego bitu badany jest również bit znajdujący się z jego prawej strony. Jeśli oba bity są takie same (1–1 lub 0–0), to wszystkie bity rejestrów A, Q i Q_{-1} są przesuwane w prawo o jeden bit. Jeśli te dwa bity różnią się, to mnożna jest dodawana lub odejmowana od zawartości rejestru A, zależnie od tego, czy bity te są 0–1, czy 1–0. Po dodawaniu lub odejmowaniu następuje przesunięcie w prawo. W każdym przypadku przesunięcie w prawo jest takie, że najdalej na lewo położony bit A, tzn. A_{n-1}, nie tylko jest przesuwany do A_{n-2}, ale także pozostaje w A_{n-1}. Jest to wymagane dla zachowania znaku liczby w A i Q. Jest to określane jako **przesunięcie arytmetyczne**, ponieważ jest zachowywany bit znaku.

A	Q	Q_{-1}	M		
0000	0011	0	0111	Stan początkowy	
1001	0011	0	0111	A ← A − M	Pierwszy cykl
1100	1001	1	0111	Przesuwanie	
1110	0100	1	0111	Przesuwanie	Drugi cykl
0101	0100	1	0111	A ← A + M	Trzeci cykl
0010	1010	0	0111	Przesuwanie	
0001	0101	0	0111	Przesuwanie	Czwarty cykl

Rysunek 9.13. Przykład algorytmu Bootha (7 × 3)

0111 ×0011 (0) 11111001 1-0 0000000 1-1 000111 0-1 00010101 (21) (a) (7) × (3) = (21)	0111 ×1101 (0) 11111001 1-0 0000111 0-1 111001 1-0 11101011 (-21) (b) (7) × (−3) = (−21)
1001 ×0011 (0) 00000111 1-0 0000000 1-1 111001 0-1 11101011 (-21) (c) (−7) × (3) = (−21)	1001 ×1101 (0) 00000111 1-0 1111001 0-1 000111 1-0 00010101 (21) (d) (−7) × (−3) = (21)

Rysunek 9.14. Przykłady zastosowania algorytmu Bootha

Na rysunku 9.13 widać sekwencję zdarzeń w algorytmie Bootha w przypadku mnożenia 7 przez 3. W sposób bardziej skondensowany ta sama operacja została przedstawiona na rys. 9.14a. W pozostałej części rys. 9.14 są podane inne przykłady stosowania tego algorytmu. Jak można zauważyć, działa on poprawnie z dowolną kombinacją liczb dodatnich i ujemnych. Zwróćmy też uwagę na efektywność tego algorytmu. Bloki jedynek lub zer są przeskakiwane, przy czym na blok przypada średnio tylko jedno dodawanie lub odejmowanie.

Dlaczego algorytm Bootha funkcjonuje? Rozważmy najpierw przypadek dodatniego mnożnika. W szczególności rozważmy dodatni mnożnik składający się z jednego bloku jedynek otoczonego przez zera, np. 00011110. Jak wiemy, mnożenie można wykonać przez sumowanie odpowiednio przesuniętych kopii mnożnej:

$$
\begin{aligned}
M \times (00011110) &= M \times (2^4 + 2^3 + 2^2 + 2^1) \\
&= M \times (16 + 8 + 4 + 2) \\
&= M \times 30
\end{aligned}
$$

Liczba takich operacji może być zredukowana do dwóch, jeśli zauważymy, że

$$2^n + 2^{n-1} + \ldots + 2^{n-K} = 2^{n+1} - 2^{n-K} \qquad (9.3)$$

Wobec tego

$$
\begin{aligned}
M \times (00011110) &= M \times (2^5 - 2^1) \\
&= M \times (32 - 2) \\
&= M \times 30
\end{aligned}
$$

Tak więc iloczyn może być generowany przez jedno dodawanie i jedno odejmowanie mnożnej. Schemat ten dotyczy dowolnej liczby bloków jedynek w mnożniku, włącznie z przypadkiem pojedynczej 1 traktowanej jako blok.

$$
\begin{aligned}
M \times (01111010) &= M \times (2^6 + 2^5 + 2^4 + 2^3 + 2^1) \\
&= M \times (2^7 - 2^3 + 2^2 - 2^1)
\end{aligned}
$$

Algorytm Bootha działa według tego właśnie schematu; wykonywane jest odejmowanie, gdy napotkana jest pierwsza 1 bloku (1–0), oraz dodawanie, gdy napotkany jest koniec bloku (0–1).

Aby wykazać, że taki sam schemat funkcjonuje w przypadku ujemnego mnożnika, musimy zwrócić uwagę na następujące zależności. Niech X będzie liczbą ujemną w notacji uzupełnienia do dwóch

Reprezentacja $X = \{1x_{n-2}x_{n-3} \ldots x_1x_0\}$

Wobec tego wartość X może być wyrażona następująco:

$$X = -2^{n-1} + (x_{n-2} \times 2^{n-2}) + (x_{n-3} \times 2^{n-3}) + \ldots + (x_1 \times 2^1) + (x_0 \times 2^0) \quad (9.4)$$

Można to zweryfikować, stosując ten algorytm do liczb w tabeli 9.2.

Najbardziej na lewo położonym bitem liczby X jest 1, ponieważ X jest ujemna. Załóżmy, że najdalej na lewo położone 0 znajduje się w pozycji k-tej. Liczba X ma więc postać

$$\text{Reprezentacja } X = \{111 \ldots 10x_{k-1}x_{k-2} \ldots x_1x_0\} \quad (9.5)$$

Wobec tego wartość X wynosi

$$X = -2^{n-1} + 2^{n-2} + \ldots + 2^{k+1} + (x_{k-1} \times 2^{k-1}) + \ldots + (x_0 \times 2^0) \quad (9.6)$$

Na podstawie równania (9.3) możemy stwierdzić, że

$$2^{n-2} + 2^{n-3} + \ldots + 2^{k+1} = 2^{n-1} - 2^{k+1}$$

Po przestawieniu

$$-2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2^{k+1} = -2^{k+1} \quad (9.7)$$

Podstawiając równanie (9.7) do (9.6), otrzymujemy

$$X = -2^{k+1} + (x_{k-1} \times 2^{k-1}) + \ldots + (x_0 \times 2^0) \quad (9.8)$$

Na zakończenie możemy wrócić do algorytmu Bootha. Jeśli weźmie się pod uwagę reprezentację X [równanie (9.5)], to staje się jasne, że wszystkie bity począwszy od X_0 aż do najdalej na lewo położonego 0 są traktowane właściwie, ponieważ umożliwiają uzyskanie wszystkich składników równania (9.8) z wyjątkiem (-2^{k+1}) i mają wobec tego właściwą postać. Gdy algorytm przegląda najdalej na lewo położone 0 i napotyka następną 1 (2^{k+1}), następuje przejście 1–0 i ma miejsce odejmowanie (-2^{k+1}). Jest to brakujący człon równania (9.8).

Jako przykład rozważmy mnożenie pewnej mnożnej przez –6. W notacji uzupełnienia do dwóch i przy zastosowaniu słowa 8-bitowego (–6) jest reprezentowane przez 11111010. Na podstawie równania (9.4) wiemy, że

$$-6 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$$

co można z łatwością sprawdzić. Wobec tego

$$M2 \times (11111010) = M \times (-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1)$$

Po zastosowaniu równania (9.7)

$$M \times (11111010) = M \times (-2^3 + 2^1)$$

co, jak można zweryfikować, nadal oznacza $M \times (-6)$. Na zakończenie, powtarzając naszą linię rozumowania, otrzymujemy

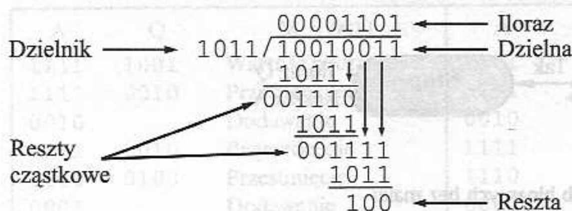
$$M \times (11111010) = M \times (-2^3 + 2^2 - 2^1)$$

Teraz możemy już stwierdzić, że algorytm Bootha działa według tego właśnie schematu. Wykonuje on odejmowanie po napotkaniu pierwszej 1 (1–0), dodawanie po napotkaniu (0–1) i wreszcie następne odejmowanie po napotkaniu pierwszej 1 z następnego bloku jedynek. Wobec tego algorytm Bootha wykonuje mniej dodawań i odejmowań niż algorytm prostszy.

Dzielenie

Dzielenie jest nieco bardziej złożone niż mnożenie, jednak opiera się na takich samych zasadach ogólnych. Jak poprzednio, podstawą algorytmu jest rozwiązanie stosowane przy obliczeniach za pomocą ołówka i papieru, a sama operacja obejmuje powtarzające się przesuwanie oraz dodawanie lub odejmowanie.

Na rysunku 9.15 jest pokazany przykład długiego dzielenia bezznakowych binarnych liczb całkowitych. Pouczające jest szczegółowe opisanie tego procesu. Po pierwsze, bity dzielnej są badane od lewej do prawej, aż zespół zbadanych bitów będzie reprezentował liczbę większą lub równą dzielnikowi; jest to określane jako sprawdzanie, czy dzielnik może *podzielić* liczbę. Aż do wystąpienia tego zdarzenia w ilorazie są umieszczane zera od lewej do prawej. Gdy wystąpi już to zdarzenie, w ilorazie umieszczana jest 1, a dzielnik jest odejmowany od dzielnej cząstkowej. Wynik jest określany jako *reszta cząstkowa*. Począwszy od tego punktu, dzielenie powtarza się cyklicznie. W każdym cyklu dodatkowe bity dzielnej są dołączane do reszty cząstkowej, aż wynik będzie większy lub równy dzielnikowi. Jak poprzednio, dzielnik jest odejmowany od tej liczby, przez co powstaje nowa reszta cząstkowa. Proces jest kontynuowany, aż wszystkie bity dzielnej zostaną zużyte.

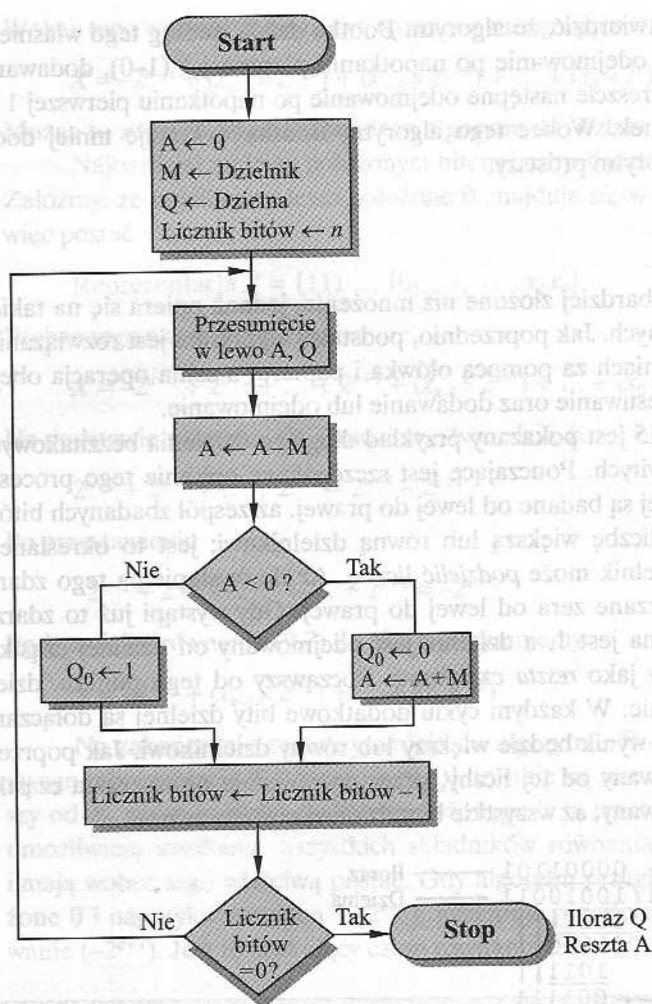


Rysunek 9.15. Dzielenie całkowitych liczb binarnych bez znaku

Na rysunku 9.16 jest pokazany algorytm maszynowy, który odpowiada procesowi dzielenia. Dzielnik jest umieszczany w rejestrze M, a dzielna w rejestrze Q. Na każdym etapie zawartości A i Q są razem przesuwane w lewo o jeden bit. Zawartość M jest odejmowana od A w celu określenia, czy A dzieli resztę cząstkową[2]. Jeśli tak, w Q_0 jest umieszczany bit 1. W przeciwnym razie Q_0 dostaje bit 0, a zawartość M musi być dodana z powrotem do zawartości A w celu odtworzenia poprzedniej wartości. Stanowi to zakończenie cyklu, a proces jest kontynuowany przez *n* cykli. Na zakończenie iloraz znajduje się w rejestrze Q, reszta zaś w rejestrze A.

---

[2] Jest to odejmowanie liczb całkowitych pozbawionych znaku. Wynik wymagający pożyczenia z najbardziej znaczącego bitu jest wynikiem ujemnym.



Rysunek 9.16. Sieć działań dzielenia liczb binarnych bez znaku

Z pewną trudnością proces ten może być rozciągnięty na liczby ujemne. Przedstawimy tu jedno z podejść odnoszące się do liczb w notacji uzupełnienia do dwóch. Kilka przykładów tego podejścia widać na rys. 9.17. Algorytm można podsumować następująco:

1. Załaduj dzielnik do rejestru M, a dzielną do rejestrów A i Q. Dzielna musi być wyrażona jako liczba w reprezentacji uzupełnienia do dwóch. Wobec tego, na przykład, 4-bitowa liczba 0111 staje się 00000111, a 1001 jest przekształcana na 111111001.
2. Przesuń A i Q w lewo o jedną pozycję bitową.
3. Jeśli M i A mają takie same znaki, przeprowadź operację A ← A − M; w przeciwnym razie A ← A + M.

4. Powyższa operacja jest udana, jeśli znak A pozostaje taki sam po zakończeniu operacji.
   (a) Jeśli operacja jest udana lub A = 0, to ustaw $Q_0 \leftarrow 1$.
   (b) Jeśli operacja jest nieudana i $A \neq 0$, to ustaw $Q_0 \leftarrow 0$ i przywróć poprzednią wartość A.
5. Powtórz kroki od 2 do 4 tyle razy, ile jest pozycji bitowych w Q.
6. Reszta jest zawarta w A. Jeśli znaki dzielnika i dzielnej były takie same, to iloraz jest w Q; w przeciwnym razie poprawny iloraz jest uzupełnieniem do dwóch zawartości rejestru Q.

A	Q	M = 0011	A	Q	M = 1101
0000	0111	Wartość początkowa	0000	0111	Wartość początkowa
0000	1110	Przesunięcie	0000	1110	Przesunięcie
1101		Odejmowanie	1101		Dodawanie
0000	1110	Przywrócenie	0000	1110	Przywrócenie
0001	1100	Przesunięcie	0001	1100	Przesunięcie
1110		Odejmowanie	1110		Dodawanie
0001	1100	Przywrócenie	0001	1100	Przywrócenie
0011	1000	Przesunięcie	0011	1000	Przesunięcie
0000		Odejmowanie	0000		Dodawanie
0000	1001	Ustawienie $Q_0 = 1$	0000	1001	Ustawienie $Q_0 = 1$
0001	0010	Przesunięcie	0001	0010	Przesunięcie
1110		Odejmowanie	1110		Dodawanie
0001	0010	Przywrócenie	0001	0010	Przywrócenie

(a) (7)/(3) (b) (7)/(–3)

A	Q	M = 0011	A	Q	M = 1101
1111	1001	Wartość początkowa	1111	1001	Wartość początkowa
1111	0010	Przesunięcie	1111	0010	Przesunięcie
0010		Dodawanie	0010		Odejmowanie
1111	0010	Przywrócenie	1111	0010	Przywrócenie
1110	0100	Przesunięcie	1110	0100	Przesunięcie
0001		Dodawanie	0001		Odejmowanie
1110	0100	Przywrócenie	1110	0100	Przywrócenie
1100	1000	Przesunięcie	1100	1000	Przesunięcie
1111		Dodawanie	1111		Odejmowanie
1111	1001	Ustawienie $Q_0 = 1$	1111	1001	Ustawienie $Q_0 = 1$
1111	0010	Przesunięcie	1111	0010	Przesunięcie
0010		Dodawanie	0010		Odejmowanie
1111	0010	Przywrócenie	1111	0010	Przywrócenie

(c) (–7)/(3) (d) (–7)/(–3)

Rysunek 9.17. Przykłady dzielenia w notacji uzupełnienia do dwóch

Na podstawie rysunku 9.17 można zauważyć, że dzieląc –7 przez 3 oraz 7 przez –3 otrzymamy różne reszty. Dzieje się tak, ponieważ reszta jest zdefiniowana następująco:

$$D = Q \times V + R$$

gdzie:
D – dzielna,
Q – iloraz,
V – dzielnik,
R – reszta.

Wyniki na rysunku 9.17 są zgodne z tym wzorem.

9.4. Reprezentacja zmiennopozycyjna

Zasady

W przypadku notacji stałopozycyjnej (np. uzupełnienia do dwóch) możliwe jest reprezentowanie zakresu dodatnich i ujemnych liczb całkowitych ze środkiem w zerze. Przy założeniu ustalonego przecinka pozycyjnego format ten umożliwia również reprezentację liczb ze składnikiem ułamkowym.

Podejście to ma ograniczenia. Nie mogą być reprezentowane ani bardzo duże liczby, ani bardzo małe ułamki. Ponadto, ułamkowe składniki ilorazu przy dzieleniu dwóch dużych liczb mogą być utracone.

W przypadku liczb dziesiętnych obchodzi się to ograniczenie stosując notację naukową. Wtedy 976 000 000 000 000 może być reprezentowana jako $9{,}76 \times 10^{14}$, a 0,0000000000000976 jako $9{,}76 \times 10^{-14}$. To, co zrobiliśmy, jest w rezultacie dynamicznym przesunięciem przecinka dziesiętnego do wygodnego położenia i użyciem potęgi 10 do określania rzeczywistego położenia tego przecinka. Umożliwia to reprezentowanie bardzo dużych i bardzo małych liczb za pomocą tylko kilku cyfr.
To samo podejście może być zastosowane do liczb binarnych. Możemy reprezentować liczbę w postaci

$$\pm S \times B^{\pm E}$$

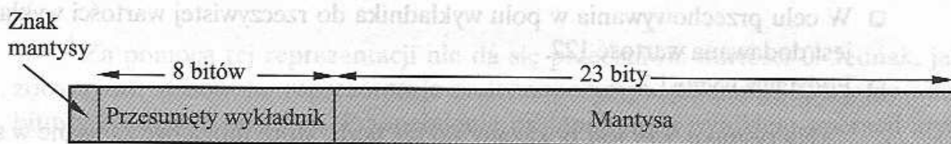
Liczba ta może być przechowywana w postaci słowa binarnego o trzech polach zawierających:

- znak: plus lub minus,
- mantysę S,
- wykładnik E.

Podstawa B jest ustalona i nie musi być przechowywana, ponieważ jest taka sama dla wszystkich liczb.

Zasady stosowane przy reprezentowaniu zmiennopozycyjnych liczb binarnych najlepiej wyjaśnić na przykładzie. Na rysunku 9.18a jest pokazany typowy, 32-bitowy format zmiennopozycyjny. Lewy bit reprezentuje znak liczby (0 – dodatnia, 1 – ujemna). Wartość **wykładnika** jest przechowywana w bitach od 1 do 8. Zastosowana tu reprezentacja jest znana jako **reprezentacja przesunięta**. Ustalona wartość, zwa-

na przesunięciem, jest odejmowana od pola w celu otrzymania prawdziwej wartości wykładnika. W tym przypadku za pomocą pola 8-bitowego można reprezentować liczby od 0 do 255. Po zastosowaniu przesunięcia równego 128 prawdziwe wartości wykładnika znajdują się w zakresie od –128 do +127. W tym przykładzie założyliśmy, że podstawa jest równa 2.



(a) Format

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$

$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$

$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$

$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$

(b) Przykłady

Rysunek 9.18. Typowy 32-bitowy format zmiennopozycyjny

Ostatnią częścią słowa (w tym przypadku 23 bity) jest **mantysa**.
Liczba zmiennopozycyjna może być wyrażana na wiele sposobów.

Następujące z nich są równoważne (mantysa jest wyrażona w postaci binarnej):

$0{,}110 \times 2^5$

110×2^2

$0{,}0110 \times 2^6$

W celu uproszczenia operacji na liczbach zmiennopozycyjnych wymagane jest zwykle, aby były one znormalizowane. **Liczbą znormalizowaną** jest taka liczba, w której najbardziej znacząca cyfra mantysy jest różna od zera. Jak już wspomniano, zgodnie z ogólnie przyjętą konwencją na lewo od przecinka znajduje się jeden bit. Zatem znormalizowana liczba niezerowa ma postać

$\pm 1{,}bbb \ldots b \times 2 \pm E$

gdzie b jest dowolną cyfrą binarną (0 lub 1). Implikuje to, że lewy bit mantysy jest zawsze równy 1. Nie jest więc konieczne przechowywanie tego bitu. Wobec tego pole 23-bitowe jest używane do przechowywania 24-bitowej mantysy o wartości znajdującej się w przedziale półotwartym [1,2). Jeśli mamy do czynienia z liczbą, która nie jest znormalizowana, można ją znormalizować, przesuwając przecinek na prawo od bitu najbardziej wysuniętego na lewo i odpowiednio dostosowując wykładnik.

Na rysunku 9.18b są podane przykłady liczb przedstawionych w tym formacie. Zwróćmy uwagę na następujące właściwości:

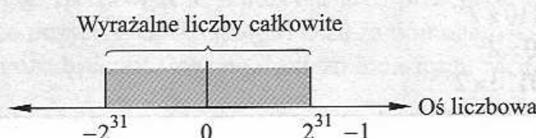
- Znak jest zawarty w pierwszym bicie słowa.
- Pierwszy bit rzeczywistej mantysy jest zawsze 1 i nie musi być przechowywany w polu znaczącym.
- W celu przechowywania w polu wykładnika do rzeczywistej wartości wykładnika jest dodawana wartość 127.
- Podstawa wynosi 2.

Na rysunku 9.19 jest pokazany zakres liczb, które mogą być zapisane w słowie 32-bitowym przy założeniu tej reprezentacji. Przy zapisaniu liczb całkowitych w notacji uzupełnienia do dwóch mogą być reprezentowane wszystkie liczby całkowite od -2^{31} do $2^{31} - 1$, co oznacza 2^{32} różnych liczb. Przyjęcie przykładowego formatu zmiennopozycyjnego z rys. 9.18 umożliwia przedstawienie liczb z następujących zakresów:

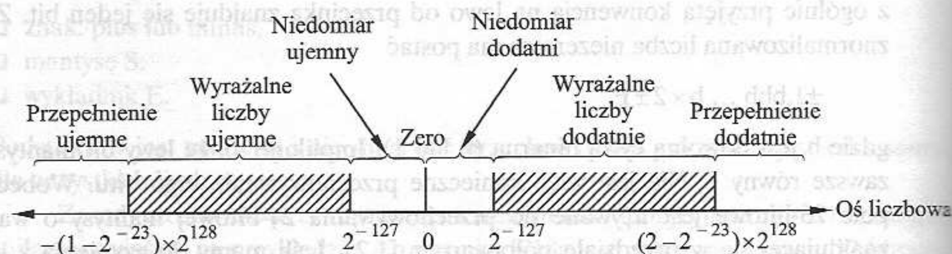
- liczby ujemne między $-(2 - 2^{-23}) \times 2^{128}$ a -2^{-127};
- liczby dodatnie między 2^{-127} a $(2 - 2^{-23}) \times 2^{128}$.

Powyższe zakresy nie obejmują pięciu obszarów:

- liczby ujemne mniejsze niż $-(2 - 2^{-23}) \times 2^{128}$, określane jako **przepełnienie ujemne**;
- liczby ujemne większe niż 2^{-127}, określane jako **niedomiar ujemny**;
- zero;
- liczby dodatnie mniejsze niż 2^{-127}, określane jako **niedomiar dodatni**;
- liczby dodatnie większe niż $(2 - 2^{-23}) \times 2^{128}$, określane jako **przepełnienie dodatnie**.

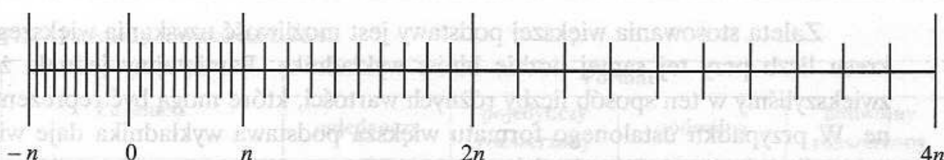


(a) Liczby całkowite w notacji uzupełnienia do dwóch



(b) Liczby zmiennopozycyjne

Rysunek 9.19. Liczby wyrażalne za pomocą typowych formatów 32-bitowych



Rysunek 9.20. Gęstość liczb zmiennopozycyjnych

Za pomocą tej reprezentacji nie da się przedstawić wartości 0. Jednak, jak zobaczymy, stosowane reprezentacje zmiennopozycyjne obejmują specjalny wzór bitowy do oznaczania zera. Przepełnienie następuje, gdy wynikiem operacji arytmetycznej jest liczba większa niż może być wyrażona przez potęgę 128 (np. $2^{120} \times 2^{100} = 2^{220}$). Niedomiar występuje natomiast wtedy, kiedy wielkość ułamkowa jest zbyt mała (np. $2^{-120} \times 2^{-100} = 2^{-220}$). Niedomiar stanowi mniej poważny problem, ponieważ wynik może być na ogół zadowalająco aproksymowany przez 0.

Ważne jest zwrócenie uwagi na to, że za pomocą notacji zmiennopozycyjnej nie reprezentuje się większej liczby pojedynczych wartości. Maksymalna liczba różnych wartości, które mogą być przedstawione za pomocą 32 bitów wynosi nadal 2^{32}. To, czego dokonaliśmy, to tylko rozproszenie tych liczb na dwa zakresy, dodatni i ujemny.

Zauważmy również, że liczby reprezentowane w notacji zmiennopozycyjnej nie są rozmieszczone równomiernie na osi liczb, jak liczby stałopozycyjne. Możliwe wartości są rozłożone gęściej na początku osi, a rzadziej w miarę oddalania się od początku, co widać na rys. 9.20. Jest to jeden z kompromisów matematyki zmiennopozycyjnej: wiele obliczeń prowadzi do wyników, które nie są dokładne i muszą być zaokrąglane do najbliższych wartości możliwych do reprezentowania za pomocą tej notacji.

W przypadku formatu pokazanego na rys. 9.18 następuje wymienność między zakresem i dokładnością. W przykładzie tym 8 bitów przeznaczono na wykładnik, a 23 na mantysę. Jeśli zwiększymy liczbę bitów wykładnika, to poszerzymy w ten sposób zakres liczb możliwych do wyrażenia. Ponieważ jednak tylko ustalona liczba różnych wartości może być wyrażona, zredukujemy w ten sposób gęstość tych liczb i tym samym dokładność. Jedyną drogą zwiększenia zarówno zakresu, jak i dokładności, jest użycie większej liczby bitów. Dlatego większość komputerów oferuje liczby w formacie o *pojedynczej* i *podwójnej precyzji*. Na przykład, format pojedynczej precyzji może mieć 32 bity, a podwójnej – 64 bity.

Występuje więc wymienność między liczbą bitów wykładnika a liczbą bitów mantysy. Jednak sprawa jest jeszcze bardziej złożona. Przyjęta domyślnie podstawa wykładnika nie musi wynosić 2. Na przykład w architekturze IBM S/370 zastosowano podstawę 16 [ANDE67b]. Format składa się z 7-bitowego wykładnika i 24-bitowej mantysy.

W formacie IBM o podstawie 16

$$0{,}11010001 \times 2^{10100} = 0{,}11010001 \times 16^{101}$$

wykładnik zaś reprezentuje liczbę 5, a nie 20.

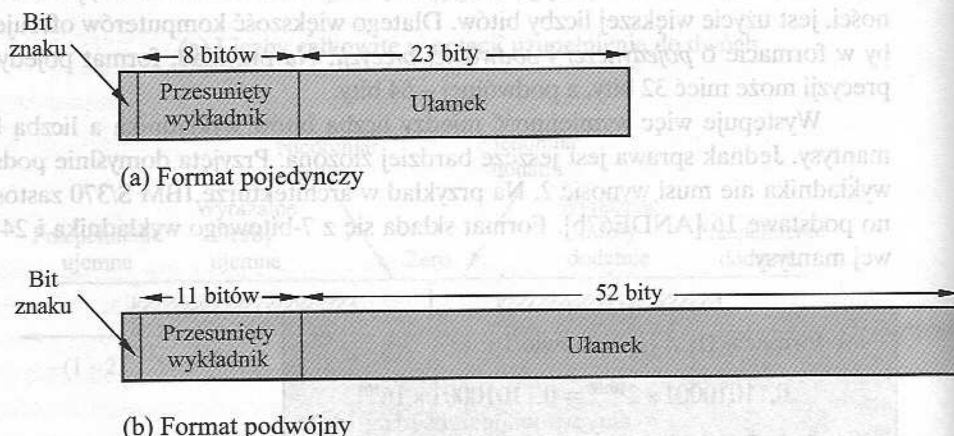
Zaletą stosowania większej podstawy jest możliwość uzyskania większego zakresu liczb przy tej samej liczbie bitów wykładnika. Pamiętajmy jednak, że nie zwiększyliśmy w ten sposób liczby różnych wartości, które mogą być reprezentowane. W przypadku ustalonego formatu większa podstawa wykładnika daje większy zakres liczb kosztem dokładności.

Norma IEEE dotycząca zmiennopozycyjnej arytmetyki binarnej

Najważniejsza reprezentacja zmiennopozycyjna jest zdefiniowana w normie IEEE Standard 754 [IEEE85]. Normę tę opracowano, aby ułatwić przenoszenie programów z jednego procesora do drugiego oraz aby pobudzić rozwój złożonych programów numerycznych. Norma ta została szeroko zaakceptowana i jest stosowana praktycznie we wszystkich współczesnych procesorach i koprocesorach arytmetycznych.

Norma IEEE określa zarówno pojedynczy format 32-bitowy, jak i podwójny 64-bitowy (rys. 9.21), z wykładnikami odpowiednio 8-bitowym i 11-bitowym. Domyślna podstawa wynosi 2. Norma definiuje ponadto dwa formaty rozszerzone, pojedynczy i podwójny, których dokładne rozwiązania zależą od implementacji. Formaty rozszerzone obejmują dodatkowe bity wykładnika (rozszerzony zakres) i mantysy (zwiększona dokładność). Są one przewidziane do stosowania w obliczeniach pośrednich. Dzięki zwiększonej dokładności, stosując rozszerzone formaty, zmniejsza się groźbę, że końcowy wynik będzie zniekształcony przez nadmierny błąd zaokrąglania; dzięki zaś większemu zakresowi, maleje groźba pośredniego przepełnienia powodującego odrzucenie obliczeń, których wynik byłby możliwy do przedstawienia w formacie podstawowym. Dodatkowym powodem stosowania formatu pojedynczego są oferowane przezeń pewne korzyści formatu podwójnego bez zwiększania czasochłonności, towarzyszącej zwykle większej dokładności. W tabeli 9.3 są przedstawione parametry tych czterech formatów.

Nie wszystkie wzory bitowe formatów IEEE są interpretowane w zwykły sposób; niektóre z nich są używane do reprezentowania wartości specjalnych. W tabeli 9.4 są



Rysunek 9.21. Formaty IEEE 754

Tabela 9.3. Parametry formatu IEEE 754

Parametr	Format			
	pojedynczy	pojedynczy rozszerzony	podwójny	podwójny rozszerzony
Długość słowa (bitów)	32	⩾43	64	⩾79
Długość wykładnika (bitów)	8	⩾11	11	⩾15
Przesunięcie wykładnika	127	nieokreślone	1023	nieokreślone
Maksymalny wykładnik	127	⩾1023	1023	⩾16383
Minimalny wykładnik	−126	⩽−1022	−1022	⩽−16382
Zakres liczb (podstawa 10)	10^{-38}, 10^{+38}	nieokreślony	10^{-308}, 10^{+308}	nieokreślony
Długość podstawy (bitów)	23	⩾31	52	⩾63
Liczba wykładników	254	nieokreślona	2046	nieokreślona
Liczba ułamków	2^{23}	nieokreślona	2^{52}	nieokreślona
Liczba wartości	$1{,}98 \times 2^{31}$	nieokreślona	$1{,}99 \times 2^{63}$	nieokreślona

podane wartości przypisane różnym wzorom bitowym. Ekstremalne wartości wykładnika w postaci samych zer (0) i samych jedynek (255 w przypadku formatu pojedynczego, 2047 w przypadku podwójnego) określają wartości specjalne. Reprezentowane są następujące klasy liczb:

- Przy wartościach wykładnika w zakresie od 1 do 254 w przypadku formatu pojedynczego oraz od 1 do 2046 w przypadku podwójnego reprezentowane są znormalizowane, niezerowe liczby zmiennopozycyjne. Wykładnik jest przesunięty, wobec tego jego zakres zawiera się od −126 do +127 w przypadku formatu pojedynczego i od −1022 do +1023 w przypadku podwójnego. Znormalizowana liczba wymaga 1 bitu na lewo od przecinka binarnego; bit ten jest domyślny, co daje efektywnie 24-bitową lub 53-bitową mantysę (określaną w normie jako *frakcja*).
- Wykładnik równy zeru w połączeniu z frakcją (mantysą) równą zeru reprezentuje dodatnie lub ujemne zero, zależnie od bitu znaku. Jak wspomnieliśmy wcześniej, użyteczne jest dokładne reprezentowanie wartości 0.
- Wykładnik złożony z samych jedynek w połączeniu z frakcją równą zeru reprezentuje dodatnią lub ujemną nieskończoność, zależnie od bitu znaku. Możliwość istnienia reprezentacji nieskończoności (∞) jest również użyteczna. Pozostawia to użytkownikowi decyzję, czy traktować przepełnienie jako warunek błędu, czy też przenieść wartość ∞ i przetwarzać ją zgodnie z programem.
- Wykładnik równy zeru w połączeniu z niezerową frakcją reprezentuje liczbę zdenormalizowaną. W tym przypadku bit na lewo od przecinka binarnego jest zerem, a rzeczywisty wykładnik jest równy −126 lub −1022. Liczba jest dodatnia lub ujemna zależnie od bitu znaku.
- Wykładnik złożony z samych jedynek w połączeniu z niezerową frakcją ma nadaną wartość NaN, która *nie oznacza liczby* (*not a number*) i jest używana do sygnalizowania różnych warunków wyjątkowych.

Znaczeniem liczb zdenormalizowanych oraz NaN zajmiemy się w podrozdz. 9.5.

Tabela 9.4. Interpretacja liczb zmiennopozycyjnych IEEE 754

	Pojedyncza precyzja (32 bity)				Podwójna precyzja (64 bity)			
	znak	przesunięty wykładnik	ułamek	wartość	znak	przesunięty wykładnik	ułamek	wartość
Zero dodatnie	0	0	0	0	0	0	0	0
Zero ujemne	1	0	0	–0	1	0	0	–0
Plus nieskończoność	0	255 (same 1)	0	∞	0	2047 (same 1)	0	∞
Minus nieskończoność	0	255 (same 1)	0	–∞	0	2047 (same 1)	0	–∞
Cicha NaN	0 lub 1	255 (same 1)	≠0	NaN	0 lub 1	2047 (same 1)	≠0	NaN
Sygnalizacyjna NaN	0 lub 1	255 (same 1)	≠0	NaN	0 lub 1	2047 (same 1)	≠0	NaN
Dodatnia znormalizowana niezerowa	0	0 < e < 255	f	2^{e-127} (1,f)	0	0 < e < 2047	f	2^{e-1023} (1,f)
Ujemna znormalizowana niezerowa	1	0 < e < 255	f	-2^{e-127} (1,f)	1	0 < e < 2047	f	-2^{e-1023} (1,f)
Dodatnia nie-znormalizowana	0	0	f≠0	2^{e-126} (0,f)	0	0	f≠0	2^{e-1022} (0,f)
Ujemna zdenormalizowana	1	0	f≠0	-2^{e-126} (0,f)	1	0	f≠0	-2^{e-1022} (0,f)

9.5. Arytmetyka zmiennopozycyjna

W tabeli 9.5 znajduje się podsumowanie podstawowych operacji arytmetyki zmiennopozycyjnej. W przypadku dodawania i odejmowania konieczne jest zapewnienie, żeby oba argumenty miały taki sam wykładnik. Może to wymagać przesunięcia przecinka pozycyjnego w jednym z argumentów. Mnożenie i dzielenie są pod tym względem prostsze.

Operacje zmiennopozycyjne mogą prowadzić do jednego z następujących wyników:

- **Przepełnienie wykładnika.** Dodatni wykładnik przekracza maksymalną dopuszczalną wartość. W pewnych systemach może to być oznaczane jako +∞ lub –∞.
- **Niedomiar wykładnika.** Ujemny wykładnik przekracza maksymalną dopuszczalną wartość (np. – 200 jest mniejsze od – 127). Oznacza to, że liczba jest zbyt mała, aby mogła być reprezentowana. Może być traktowana jako 0.
- **Niedomiar mantysy.** W procesie wyrównywania mantys cyfry mogą „wypłynąć" poza prawy koniec mantysy. Wymagana jest pewna forma zaokrąglenia, co jeszcze przedyskutujemy.

- **Przepełnienie mantysy.** Wynikiem dodawania mantys o takim samym znaku może być wyprowadzenie najbardziej znaczącego bitu. Można to naprawić przez powtórne wyrównywanie, co jeszcze wyjaśnimy.

Tabela 9.5. Liczby zmiennopozycyjne i operacje arytmetyczne

Liczby zmiennopozycyjne	Operacje arytmetyczne
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$\left.\begin{array}{l} X + Y = (X_S B^{X_E - Y_E} + Y_S) \times B^{Y_E} \\ X - Y = (X_S B^{X_E - Y_E} - Y_S) \times B^{Y_E} \end{array}\right\} X_E \leqslant Y_E$ $X \times Y = (X_S \times Y_S) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

Przykłady:
$X = 0{,}3 \times 10^2 = 30$
$Y = 0{,}2 \times 10^3 = 200$
$X + Y = (0{,}3 \times 10^{2-3} + 0{,}2) \times 10^3 = 0{,}23 \times 10^3 \times 10^3 = 230$
$X - Y = (0{,}3 \times 10^{2-3} - 0{,}2) \times 10^3 = (-0{,}17) \times 10^3 = -170$
$X \times Y = (0{,}3 \times 0{,}2) \times 10^{2+3} = 0{,}06 \times 10^5 = 6000$
$X \div Y = (0{,}3 \div 0{,}2) \times 10^{2-3} = 1{,}5 \times 10^{-1} = 0{,}15$

Dodawanie i odejmowanie

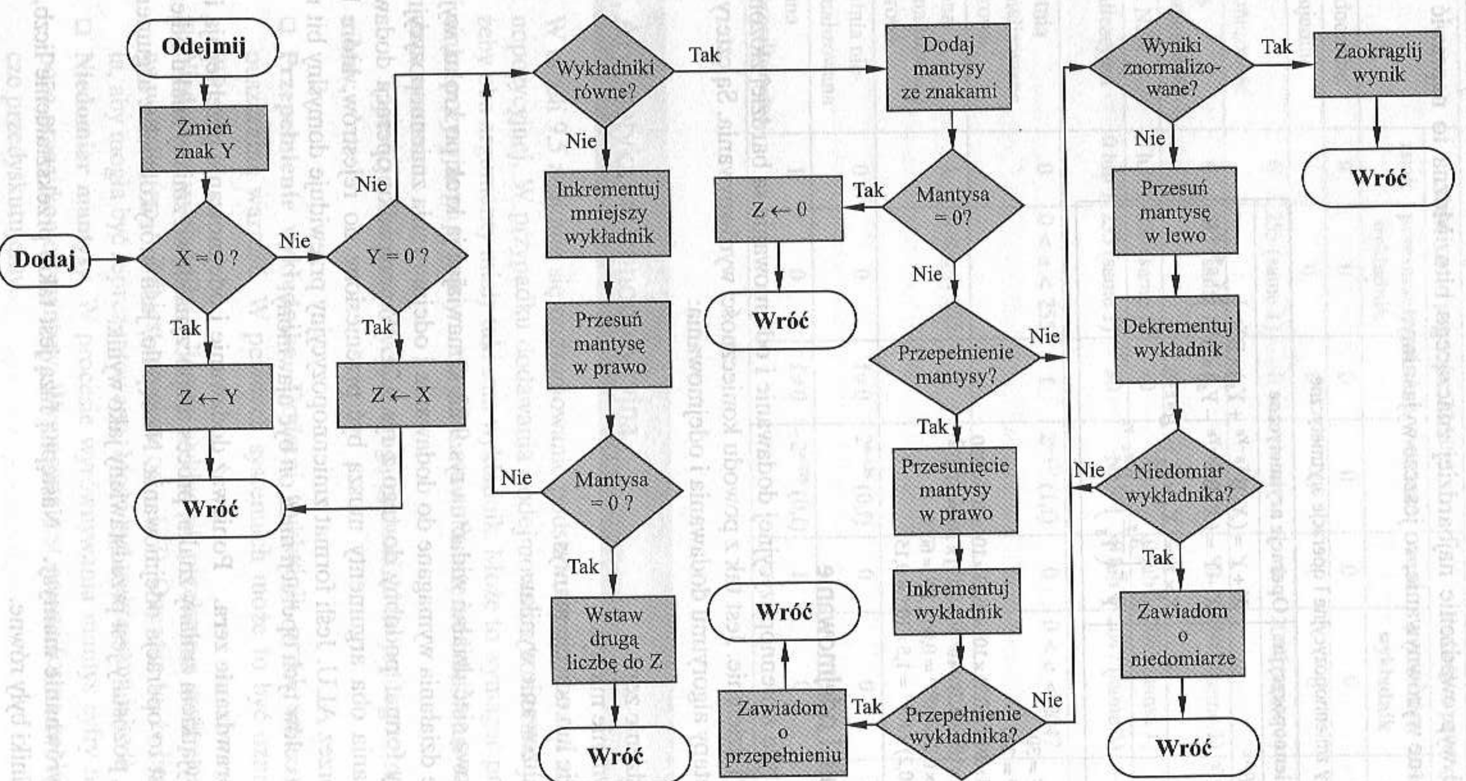
W arytmetyce zmiennopozycyjnej dodawanie i odejmowanie są bardziej złożone niż mnożenie i dzielenie. Jest tak z powodu konieczności wyrównywania. Są cztery podstawowe etapy algorytmu dodawania i odejmowania:

1. Sprawdzenie zer.
2. Wyrównanie mantys.
3. Dodanie lub odjęcie mantys.
4. Normalizowanie wyniku.

Typową sieć działań widać na rys. 9.22. Omawiając ją krok po kroku, wyjaśnimy główne działania wymagane do dodawania i odejmowania zmiennopozycyjnego. Zakładamy format podobny do tego z rys. 9.21. W przypadku operacji dodawania i odejmowania oba argumenty muszą być przeniesione do rejestrów, które będą używane przez ALU. Jeśli format zmiennopozycyjny przewiduje domyślny bit mantysy, to dla celów tych operacji musi on być ujawniony.

Faza 1. Sprawdzanie zera. Ponieważ dodawanie i odejmowanie przebiegają identycznie z wyjątkiem zmiany znaku, proces rozpoczyna się od zmiany znaku odjemnika, jeśli jest to operacja odejmowania. Następnie, jeśli którykolwiek z argumentów jest zerem, pozostały jest przedstawiany jako wynik.

Faza 2. Wyrównanie mantysy. Następną fazą jest takie przekształcenie liczb, aby oba wykładniki były równe.



Rysunek 9.22. Dodawanie i odejmowanie zmiennopozycyjne (Z ← X ± Y)

Żeby dostrzec potrzebę tego przekształcenia, rozważmy następujące dodawanie dziesiętne:

$$(123 \times 10^0) + (456 \times 10^{-2})$$

Jest oczywiste, że nie możemy po prostu dodać mantys. Cyfry muszą najpierw być ustawione w pozycjach równoważnych, to znaczy 4 z drugiej liczby musi być *wyrównana* z 3 z pierwszej. Wtedy oba wykładniki będą równe, co jest matematycznym warunkiem dodawania dwóch liczb w tej postaci. Wobec tego

$$(123 \times 10^0) + (456 \times 10^{-2}) = (123 \times 10^0) + (4{,}56 \times 10^0) = 127{,}56 \times 10^0$$

Wyrównanie jest osiągane przez przesuwanie albo mniejszej liczby w prawo (zwiększanie jej wykładnika), albo większej liczby w lewo. Ponieważ wynikiem którejkolwiek z tych operacji może być utrata cyfr, przesuwa się raczej mniejszą liczbę; ewentualne stracone cyfry mają stosunkowo małe znaczenie. Wyrównanie jest osiągane przez powtarzające się przesuwanie mantysy o jedną cyfrę w prawo i odpowiednie zwiększanie wykładnika, aż do zrównania się wykładników. (Zauważmy, że jeśli podstawa jest równa 16, to przesunięcie o jedną cyfrę jest przesunięciem o 4 bity). Jeśli wynikiem procesu jest zerowa wartość mantysy, to pozostała liczba jest traktowana jako wynik. Jeśli więc obie liczby mają znacznie różniące się wykładniki, mniejsza liczba jest tracona.

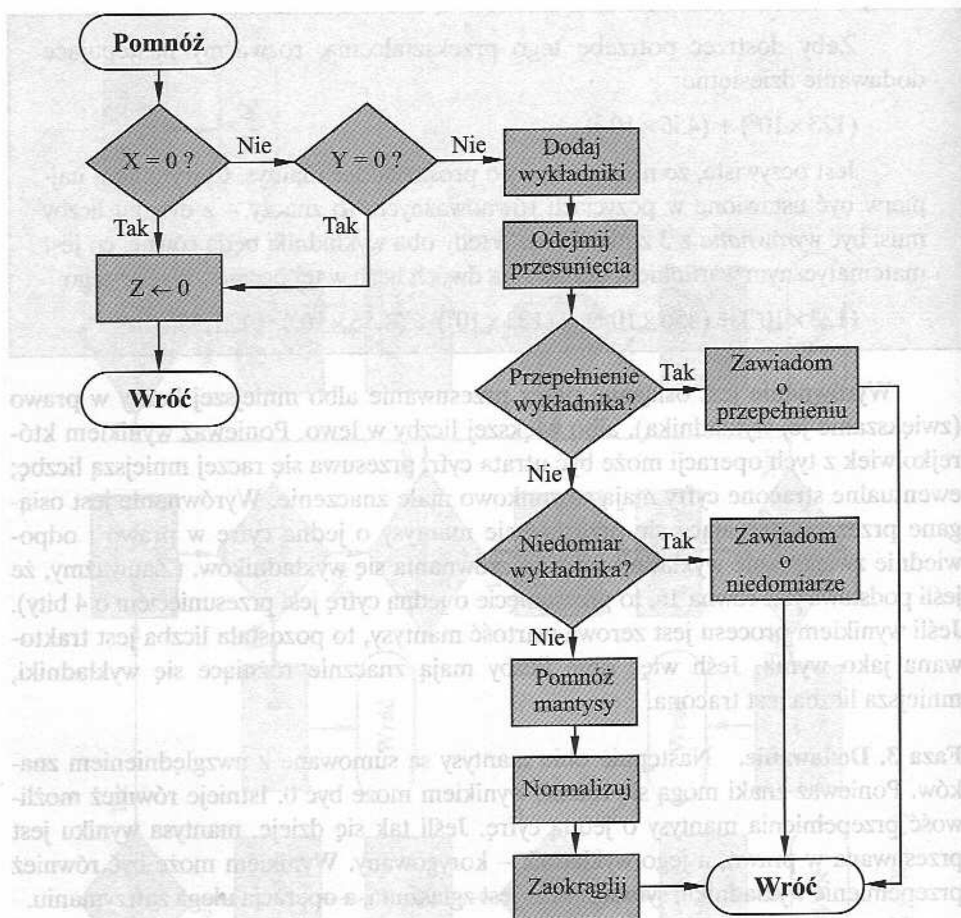
Faza 3. Dodawanie. Następnie obie mantysy są sumowane z uwzględnieniem znaków. Ponieważ znaki mogą się różnić, wynikiem może być 0. Istnieje również możliwość przepełnienia mantysy o jedną cyfrę. Jeśli tak się dzieje, mantysa wyniku jest przesuwana w prawo, a jego wykładnik – korygowany. Wynikiem może być również przepełnienie wykładnika; sytuacja taka jest zgłaszana, a operacja ulega zatrzymaniu.

Faza 4. Normalizacja. Następną fazą jest normalizowanie wyniku. Normalizacja polega na przesuwaniu cyfr mantysy w lewo, aż najbardziej znacząca cyfra (bit lub 4 bity przy podstawie 16) nie będzie zerem. Każde przesunięcie powoduje zmniejszenie wykładnika i dlatego może być przyczyną niedomiaru wykładnika. Na zakończenie wynik musi być zaokrąglony i zgłoszony. Dyskusję zaokrąglania przeprowadzimy po omówieniu mnożenia i dzielenia.

Mnożenie i dzielenie

Mnożenie i dzielenie zmiennopozycyjne są procesami o wiele prostszymi niż dodawanie i odejmowanie, co wyniknie z dalszej dyskusji.

Rozważmy mnożenie zilustrowane na rys. 9.23. Najpierw, jeśli którykolwiek z argumentów jest równy 0, jako wynik jest zgłaszane 0. Następnym krokiem jest dodanie wykładników. Jeśli wykładniki są przechowywane w postaci przesuniętej, to suma wykładników spowodowałaby podwojenie przesunięcia. Wobec tego wartość przesunięcia musi być odjęta od sumy. Wynikiem może być przepełnienie lub niedomiar wykładnika, co powinno być zgłoszone i co stanowi zakończenie algorytmu.

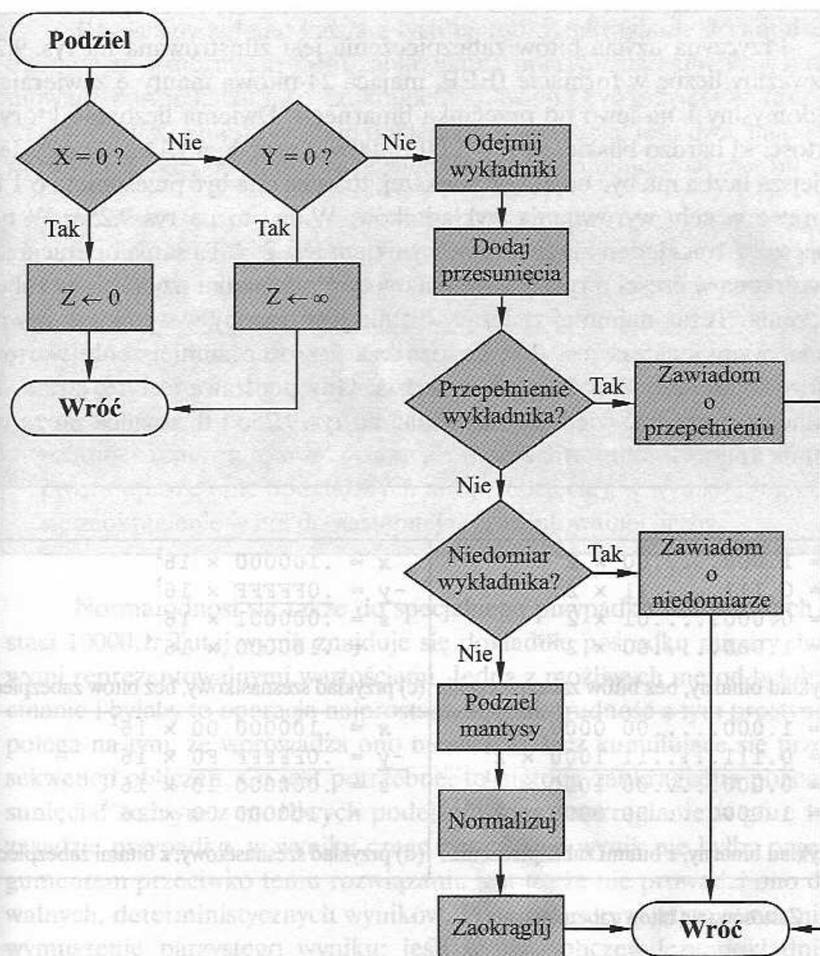


Rysunek 9.23. Mnożenie zmiennopozycyjne ($Z \leftarrow X \times Y$)

Jeśli wykładnik iloczynu jest we właściwym zakresie, to następnym krokiem jest mnożenie mantys z uwzględnieniem ich znaków. Mnożenie jest realizowane w ten sam sposób, jak w przypadku liczb całkowitych. Mamy tu do czynienia z reprezentacją znak-moduł, jednak szczegóły są podobne do mnożenia liczb wyrażonych w notacji uzupełnienia do dwóch. Iloczyn ma długość dwukrotnie większą od długości mnożnej i mnożnika. Dodatkowe bity będą utracone podczas zaokrąglania.

Po obliczeniu iloczynu wynik jest normalizowany i zaokrąglany, podobnie jak przy dodawaniu i odejmowaniu. Zauważmy, że normalizacja może spowodować niedobór wykładnika.

Na zakończenie rozważmy sieć działań dla dzielenia pokazaną na rys. 9.24. Znów pierwszym krokiem jest sprawdzanie zer. Jeśli dzielnik jest równy 0, to jest zgłaszany błąd lub wynik jest ustalany jako nieskończoność, zależnie od implementacji. Dzielna równa 0 daje wynik 0. Następnie wykładnik dzielnika jest odejmowany od wykładnika dzielnej. Powoduje to usunięcie przesunięcia, które wobec tego musi być dodane. Wykonywane są testy w celu wykrycia przepełnienia lub niedomiaru wykładnika.



Rysunek 9.24. Dzielenie zmiennopozycyjne (Z ← X/Y)

Następnym krokiem jest dzielenie mantys. Po tym następuje zwykła normalizacja i zaokrąglanie.

Analiza dokładności

Bity zabezpieczenia

Wspomnieliśmy, że przed operacją zmiennopozycyjną wykładnik i mantysa każdego argumentu są ładowane do rejestrów ALU. W przypadku mantysy długość rejestru jest prawie zawsze większa niż długość mantysy wraz z bitem domyślnym, jeśli taki występuje. Rejestr zawiera dodatkowe bity, zwane **bitami zabezpieczenia**, które są używane do wypełniania prawej strony mantysy zerami.

Przyczyna użycia bitów zabezpieczenia jest zilustrowana na rys. 9.25. Rozważmy liczbę w formacie IEEE, mającą 24-bitową mantysę zawierającą bit domyślny 1 na lewo od przecinka binarnego. Dwiema liczbami, których wartości są bardzo bliskie, są $X = 1{,}00 \ldots 00 \times 2^1$ oraz $Y = 1{,}11 \ldots 11 \times 2^0$. Jeśli mniejsza liczba ma być odjęta od większej, to musi ona być przesunięta o 1 bit w prawo w celu wyrównania wykładników. Widać to na rys. 9.25a. W tym procesie Y traci jeden bit znaczący; wynikiem jest 2^{-22}. Ta sama operacja jest powtórzona w części b rysunku, jednak w tym przypadku użyto bitów zabezpieczenia. Teraz najmniej znaczący bit nie jest tracony w wyniku wyrównywania, wynikiem zaś jest 2^{-23}, co oznacza 2-krotne zmniejszenie wartości w porównaniu z poprzednią odpowiedzią. Gdy podstawą jest 16, strata dokładności może być większa. Jak widać na rys. 9.25c i d, różnicą może być czynnik 16.

x = 1.000.....00 × 2^{1} $-y$ = 0.111.....11 × 2^{1} z = 0.000.....01 × 2^{1} = 1.000.....00 × 2^{-22}	x = .100000 × 16^{1} $-y$ = .0FFFFF × 16^{1} z = .000001 × 16^{1} = .100000 × 16^{-4}
(a) przykład binarny, bez bitów zabezpieczenia	(c) przykład szesnastkowy, bez bitów zabezpieczenia
x = 1.000.....00 0000 × 2^{1} $-y$ = 0.111.....11 1000 × 2^{1} z = 0.000.....00 1000 × 2^{1} = 1.000.....00 0000 × 2^{-23}	x = .100000 00 × 16^{1} $-y$ = .0FFFFF F0 × 16^{1} z = .000000 10 × 16^{1} = .100000 00 × 16^{-5}
(b) przykład binarny, z bitami zabezpieczenia	(d) przykład szesnastkowy, z bitami zabezpieczenia

Rysunek 9.25. Zastosowanie bitów zabezpieczenia

Zaokrąglanie

Innym czynnikiem wpływającym na dokładność wyniku jest metoda zaokrąglania. Wynik jakiejkolwiek operacji na mantysach jest na ogół przechowywany w dłuższym rejestrze. Podczas sprowadzania wyniku do formatu zmiennopozycyjnego dodatkowe bity muszą być usunięte.

Zbadano wiele metod zaokrąglania. Norma IEEE wymienia cztery alternatywne podejścia:

- **Zaokrąglanie do najbliższej.** Wynik jest zaokrąglany do najbliższej reprezentowalnej liczby.
- **Zaokrąglanie w kierunku $+\infty$.** Wynik jest zaokrąglany w górę, w kierunku plus nieskończoności.
- **Zaokrąglanie w kierunku $-\infty$.** Wynik jest zaokrąglany w dół, w kierunku minus nieskończoności.
- **Zaokrąglanie w kierunku 0.** Wynik jest zaokrąglany w kierunku zera.

Rozważmy kolejno każdą z tych metod. **Zaokrąglanie do najbliższej** jest wymienionym w normie trybem zaokrąglania opartym na zaniedbywaniu i jest definiowane następująco: jako zaokrąglenie powinna być przyjęta wartość reprezentowalna najbliższa nieskończenie dokładnemu wynikowi; jeśli dwie najbliższe wartości są równoodległe, powinna być przyjęta ta, której najmniej znaczący bit jest zerem.

> Jeśli dodatkowe bity, poza 23 bitami możliwymi do przechowania, są równe 10010, to dodatkowe bity dają w wyniku więcej niż połowę ostatniej reprezentowalnej pozycji bitowej. W tym przypadku prawidłową odpowiedzią jest dodanie binarnej 1 do ostatniego reprezentowalnego bitu, co stanowi zaokrąglenie do następnej reprezentowalnej liczby. Przyjmijmy teraz, że dodatkowymi bitami są 01111; wtedy dodatkowe bity stanowią mniej niż połowę ostatniej reprezentowalnej pozycji bitowej. Prawidłową odpowiedzią jest po prostu opuszczenie dodatkowych bitów (obcięcie), w wyniku czego otrzymuje się zaokrąglenie w dół do następnej reprezentowalnej liczby.

Norma odnosi się także do specjalnego przypadku dodatkowych bitów w postaci 10000... Tutaj wynik znajduje się dokładnie pośrodku między dwiema możliwymi reprezentowalnymi wartościami. Jedną z możliwych metod byłoby zawsze obcinanie i byłaby to operacja najprostsza. Jednak trudność z tym prostym podejściem polega na tym, że wprowadza ono niewielkie, lecz kumulujące się przesunięcie do sekwencji obliczeń. Co jest potrzebne, to metoda zaokrąglania pozbawiona przesunięcia. Jednym z możliwych podejść byłoby zaokrąglanie w górę lub w dół na zasadzie przypadku, w wyniku czego przeciętnie wynik nie byłby przesunięty. Argumentem przeciwko temu rozwiązaniu jest to, że nie prowadzi ono do przewidywalnych, deterministycznych wyników. Podejściem przyjętym w normie IEEE jest wymuszenie parzystego wyniku: jeśli wynik obliczeń leży dokładnie pośrodku między dwiema reprezentowalnymi liczbami, to wartość jest zaokrąglana w górę, gdy ostatnim reprezentowalnym bitem jest 1, lub jest pozostawiana bez zmiany, jeśli jest nim 0.

Dwie następne możliwości, **zaokrąglanie do plus i minus nieskończoności**, są użyteczne przy wdrażaniu metody znanej jako *arytmetyka przedziałów* (*interval arithmetic*). Pomysł będący podstawą arytmetyki przedziałów jest następujący. Po zakończeniu sekwencji operacji zmiennopozycyjnych nie możemy znać dokładnej odpowiedzi ze względu na ograniczenia sprzętowe wprowadzające zaokrąglanie. Jeśli przeprowadzimy każde obliczenie w sekwencji dwukrotnie, za jednym razem zaokrąglając w górę, a za drugim w dół, w rezultacie otrzymamy górną i dolną granicę dokładnej odpowiedzi. Jeśli odstęp między górną a dolną granicą jest odpowiednio mały, to uzyskamy odpowiednio dokładny wynik. Jeśli nie, to przynajmniej o tym wiemy i możemy przeprowadzić dodatkową analizę.

Ostatnią metodą wymienioną w normie jest **zaokrąglanie w kierunku zera**. Jest to w rzeczywistości po prostu obcinanie: dodatkowe bity są ignorowane. Jest to z pewnością metoda najprostsza. Jednak jej rezultatem jest to, że wielkość obciętej

liczby jest zawsze mniejsza lub równa dokładniejszej liczbie oryginalnej, co wprowadza ciągłe przesunięcie w dół. Jest to większe przesunięcie niż omawiane wyżej, ponieważ dotyczy ono każdej operacji mającej niezerowe bity dodatkowe.

Norma IEEE dotycząca binarnej arytmetyki zmiennopozycyjnej

Norma IEEE 754 wykracza poza prostą definicję formatu, określając specyficzne metody i procedury służące do tego, aby arytmetyka zmiennopozycyjna dawała jednolite i przewidywalne wyniki, niezależne od platformy sprzętowej. Jeden z aspektów już omówiliśmy, mianowicie zaokrąglanie. Przedmiotem tego punktu są trzy inne zagadnienia: nieskończoność, symbole NaN i liczby zdenormalizowane.

Nieskończoność

Arytmetyka nieskończoności jest traktowana jako przypadek graniczny rzeczywistej arytmetyki, przy czym wartościom nieskończoności jest nadana następująca interpretacja:

$$-\infty < (\text{każda skończona liczba}) < +\infty$$

Z wyjątkiem szczególnych przypadków przedyskutowanych poniżej, każda operacja arytmetyczna obejmująca nieskończoność daje oczywiste wyniki.

Na przykład

$5 + (+\infty) = +\infty$	$5 \div (+\infty) = +0$
$5 - (+\infty) = -\infty$	$(+\infty) + (+\infty) = +\infty$
$5 + (-\infty) = -\infty$	$(-\infty) + (-\infty) = -\infty$
$5 - (-\infty) = +\infty$	$(-\infty) - (+\infty) = -\infty$
$5 \times (+\infty) = +\infty$	$(+\infty) - (-\infty) = +\infty$

Ciche i sygnalizacyjne jednostki NaN

NaN jest symboliczną jednostką zakodowaną w formacie zmiennopozycyjnym, która może występować w dwóch rodzajach: cichym i sygnalizacyjnym. *Sygnalizacyjna NaN* służy do sygnalizowania wyjątków nieważnej operacji, gdy pojawia się ona jako jeden z argumentów. *Sygnalizacyjna NaN* nadaje wartość niezainicjowanym zmiennym oraz rozszerzeniom quasiarytmetycznym, które nie są przedmiotem normy. *Cicha NaN* propaguje się przez prawie wszystkie operacje arytmetyczne, nie sygnalizując wyjątku. W tabeli 9.6 są pokazane operacje powodujące powstanie cichych NaN.

Zauważmy, że oba rodzaje NaN mają taki sam format ogólny (tabela 9.4): wykładnik złożony z samych jedynek i niezerowa frakcja. Rzeczywisty wzór bitowy

frakcji niezerowej jest zależny od implementacji; wartości frakcji mogą być używane do odróżniania cichych NaN od sygnalizacyjnych oraz do określania szczególnych warunków wyjątkowych.

Tabela 9.6. Operacje prowadzące do cichej NaN

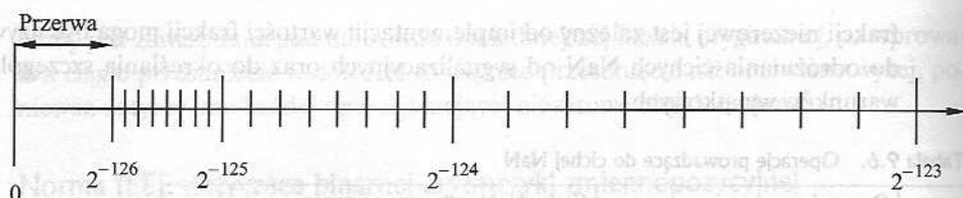
Operacja	**Cicha NaN wytworzona przez**
Dowolna	dowolną operację na sygnalizacyjnej NaN
Dodawanie lub odejmowanie	operacje na nieskończonościach: $(+\infty)+(-\infty)$ $(-\infty)+(+\infty)$ $(+\infty)-(+\infty)$ $(-\infty)-(-\infty)$
Mnożenie	$0\times\infty$
Dzielenie	$\frac{0}{0}$ lub $\frac{\infty}{\infty}$
Reszta	x reszta 0 lub ∞ reszta y
Pierwiastek kwadratowy	$\sqrt{x}$, gdzie $x < 0$

Liczby zdenormalizowane

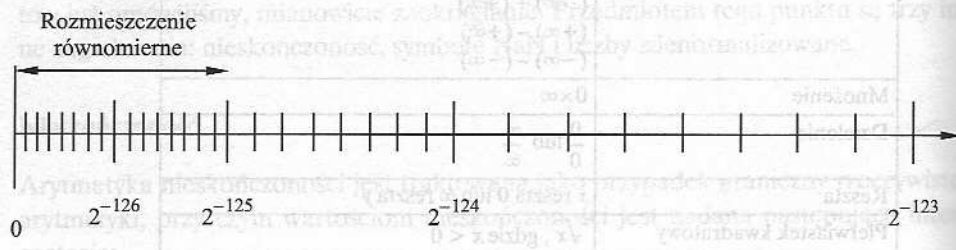
Liczby zdenormalizowane są włączone do normy IEEE 754 w celu użycia w razie niedomiaru wykładnika. Gdy wykładnik wyniku jest zbyt mały (w przypadku ujemnego wykładnika ma zbyt dużą wielkość), wynik jest denormalizowany przez przesuwanie frakcji w prawo i zwiększanie wykładnika za każdym przesunięciem, aż wykładnik znajdzie się w zakresie reprezentowalnym.

Na rysunku 9.26 jest pokazany efekt wykorzystania liczb zdenormalizowanych. Liczby reprezentowalne mogą być pogrupowane w przedziały o postaci $[2^n, 2^{n+1}]$. Wewnątrz każdego takiego przedziału wykładniki pozostają stałe, a frakcje zmieniają się, tworząc jednolite rozmieszczenie reprezentowalnych liczb wewnątrz przedziału. Gdy zbliżamy się do zera, każdy kolejny przedział ma szerokość równą połowie szerokości poprzedniego przedziału, jednak zawiera taką samą ilość reprezentowalnych liczb. Wobec tego gęstość reprezentowalnych liczb wzrasta w miarę zbliżania się do zera. Jeżeli jednak używamy wyłącznie liczb znormalizowanych, istnieje przerwa między najmniejszą liczbą znormalizowaną a zerem. W przypadku 32-bitowego formatu IEEE 754 w każdym przedziale istnieje 2^{23} reprezentowalnych liczb, a najmniejszą reprezentowalną liczbą dodatnią jest 2^{-126}. Dzięki dodaniu liczb zdenormalizowanych w przedziale między 0 a 2^{-126} uzyskuje się dodatkowe, równomiernie rozmieszczone 2^{23} liczb.

Stosowanie liczb zdenormalizowanych jest określane jako *stopniowy niedomiar* (*gradual underflow*) [COON81]. Bez liczb zdenormalizowanych przerwa między najmniejszą reprezentowalną liczbą niezerową a zerem jest o wiele szersza niż przerwa między najmniejszą reprezentowalną liczbą niezerową a następną większą liczbą. Stopniowy niedomiar powoduje wypełnienie tej przerwy i zredukowanie wpływu niedomiaru wykładnika do poziomu porównywalnego z zaokrąglaniem liczb znormalizowanych.



(a) Format 32-bitowy bez liczb zdenormalizowanych



(b) Format 32-bitowy z liczbami zdenormalizowanymi

Rysunek 9.26. Wpływ liczb zdenormalizowanych IEEE 754

9.6. Polecana literatura i witryny WWW

[PARH00] to doskonałe ujęcie arytmetyki komputerowej, obejmujące szczegółowo wszystkie zagadnienia poruszone w tym rozdziale. [FLYN01] to przydatna analiza koncentrująca się na zagadnieniach praktycznego projektowania i implementacji. Bardzo użytecznym źródłem dla poważnych studentów arytmetyki komputerowej jest dwutomowa książka [SWAR90]. Tom I został opublikowany po raz pierwszy w roku 1980 i zawiera podstawowe artykuły (niektóre trudne do uzyskania inną drogą) poświęcone podstawom arytmetyki komputerowej. Tom II zawiera nowsze artykuły dotyczące teoretycznych, projektowych i implementacyjnych aspektów arytmetyki komputerowej.

Jeśli chodzi o arytmetykę zmiennopozycyjną, dobrze nazwana jest publikacja [GOLD91]: „Co każdy informatyk powinien wiedzieć o arytmetyce zmiennopozycyjnej". Inne doskonałe ujęcie tego tematu znajduje się w [KNUT02]; obejmuje ono również całkowitoliczbową arytmetykę komputerową. Wartościowe są także następujące, wyróżniające się głębokością ujęcia publikacje: [OVER01, EVEN00, OBER97a, OBER97b, SODE96].

W [SCHW99] opisano pierwszy procesor IBM S/390 integrujący arytmetykę zmiennopozycyjną o podstawie 16 oraz opartą na normie IEEE 754 w tej samej jednostce arytmetyczno-logicznej.

EVEN00 Even G., Paul W.: „On the Design of IEEE Compliant Floating-Point Units". *IEEE Transactions on Computers*, May 2000.

FLYN01 Flynn M., Oberman S.: *Advanced Computer Arithmetic Design*. New York, Wiley, 2001.

GOLD91 Goldberg D.: „What Every Computer Scientist Should Know About Floating-Point Arithmetic". *ACM Computing Surveys*, March 1991. Dostępne w http://www.validgh.com

KNUT02 Knuth D.: *Sztuka programowania, Tom 2: Algorytmy seminumeryczne*. WNT, Warszawa 2002.

OBER97a Oberman S., Flynn M.: „Design Issues in Division and Other Floating-Point Operations". *IEEE Transactions on Computers*, February 1997.

OBER97b Oberman S., Flynn M.: „Division Algorithms and Implementations". *IEEE Transactions on Computers*, August 1997.

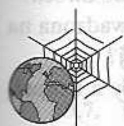
OVER01 Overton M.: *Numerical Computing with IEEE Floating Point Arithmetic*. Philadelphia, Society for Industrial and Applied Mathematics, 2001.

PARH00 Parhami B.: *Computer Arithmetic: Algorithms and Hardware Design*. Oxford, Oxford University Press, 2000.

SCHW99 Schwarz E., Krygowski C.: „The S/390 G5 Floating-Point Unit". *IBM Journal of Research and Development*, September/November 1999 (WWW).

SODE96 Soderquist P., Leeser M.: „Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementations". *ACM Computing Surveys*, September 1996.

SWAR90 Swartzlander E. (ed.): *Computer Arithmetic, Volumes I and II*. Los Alamitos, IEEE Computer Society Press, 1990.


Polecana witryna WWW:

❑ **IEEE 754.** Dokumentacja IEEE 754, dotycząca artykułów i publikacji oraz użyteczny zbiór łączy związanych z arytmetyką komputerową.

9.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Bit znaku – *sign bit*
Bity ochronne – *guard bits*
Dzielna – *dividend*
Dzielnik – *divisor*
Iloczyn – *product*
Iloczyn cząstkowy – *partial product*
Iloraz – *quotient*
Jednostka arytmetyczno-logiczna (ALU) – *arithmetic and logic unit*
Liczba zdenormalizowana – *denormalized number*
Liczba znormalizowana – *normalized number*
Mantysa – *significand*
Mnożna – *multiplicand*
Mnożnik – *multiplier*
Niedopełnienie dodatnie – *positive underflow*
Niedopełnienie mantysy – *significand underflow*
Niedopełnienie ujemne – *negative underflow*
Niedopełnienie wykładnika – *exponent underflow*
Odjemna – *minuend*
Odjemnik – *subtrahend*
Podstawa – *radix*
Przepełnienie – *overflow*
Przepełnienie dodatnie – *positive overflow*
Przepełnienie mantysy – *significand overflow*
Przepełnienie ujemne – *negative overflow*
Przepełnienie wykładnika – *exponent overflow*
Przesunięcie arytmetyczne – *arithmetic shift*
Reprezentacja dopełnienia do jedności – *ones-complement representation*

Reprezentacja moduł-znak – *sign-magnitude representation*

Reprezentacja przesunięta – *biased representation*

Reprezentacja stałopozycyjna – *fixed point representation*

Reprezentacja uzupełnienia do dwóch – *two's complement representation*

Reprezentacja zmiennopozycyjna – *floating-point representation*

Reszta – *remainder*

Separator pozycyjny – *radix point*

Wykładnik – *exponent*

Zaokrąglenie – *rounding*

Pytania kontrolne

9.1. Krótko objaśnij następujące reprezentacje: moduł-znak, uzupełnienie do dwóch, przesuniętą.

9.2. Wyjaśnij, jak stwierdzić, czy liczba jest ujemna w następujących reprezentacjach: moduł-znak, uzupełnienie do dwóch, przesuniętej.

9.3. Jaka jest reguła poszerzania znaku w odniesieniu do liczb wyrażonych w postaci uzupełnienia do dwóch?

9.4. Jak można zrealizować negację liczby całkowitej w reprezentacji uzupełnienia do dwóch?

9.5. Kiedy – w kategoriach ogólnych – operacja uzupełnienia do dwóch przeprowadzona na n-bitowej liczbie całkowitej prowadzi do uzyskania tej samej liczby całkowitej?

9.6. Jaka jest różnica między reprezentacją uzupełnienia do dwóch jakiejś liczby a uzupełnieniem do dwóch tej liczby?

9.7. Jeśli do celów dodawania potraktujemy dwie liczby wyrażone w notacji uzupełnienia do dwóch jako bezznakowe liczby całkowite, wynik będzie poprawny, gdy zostanie zinterpretowany jako liczba w notacji uzupełnienia do dwóch. W przypadku mnożenia tak nie jest. Dlaczego?

9.8. Jakie są cztery podstawowe elementy liczby w notacji zmiennopozycyjnej?

9.9. Jaka jest korzyść ze stosowania reprezentacji przesuniętej w odniesieniu do wykładnika liczby zmiennopozycyjnej?

9.10. Jakie są różnice między przepełnieniem dodatnim, przepełnieniem wykładnika oraz przepełnieniem mantysy?

9.11. Jakie są podstawowe elementy dodawania i odejmowania zmiennopozycyjnego?

9.12. Podaj powód używania bitów zabezpieczenia.

9.13. Wymień alternatywne metody zaokrąglania wyniku operacji zmiennopozycyjnej.

Problemy do rozwiązania

9.1. Inną, spotykaną czasem reprezentacją binarnych liczb całkowitych jest **uzupełnienie jedynkowe.** Dodatnie liczby całkowite są reprezentowane w taki sposób jak wielkość znaku. Ujemne liczby całkowite są reprezentowane przez wykorzystanie uzupełnienia boolowskiego każdego bitu odpowiedniej liczby dodatniej.

(a) Podaj definicję liczb wyrażonych w postaci uzupełnienia jedynkowego, posługując się ważoną sumą bitów, podobnie jak w równaniach (9.1) i (9.2).

(b) Jaki jest zakres liczb, które mogą być reprezentowane za pomocą uzupełnienia jedynkowego?

(c) Podaj algorytm dodawania w arytmetyce uzupełnienia jedynkowego.

9.2. Do tabeli 9.1 dodaj kolumny wielkości znaku i uzupełnienia jedynkowego.

9.3. Rozważ następującą operację na słowie binarnym. Rozpocznij od najmniej znaczącego bitu. Kopiuj wszystkie bity będące zerami, aż napotkasz pierwszy bit, który również skopiuj. Następnie oblicz uzupełnienie każdego z pozostałych bitów. Jaki jest wynik?

9.4. W podrozdziale 9.3 operacja uzupełniania do dwóch jest zdefiniowana następująco. Aby znaleźć uzupełnienie do dwóch X, weź uzupełnienie boolowskie każdego bitu X, po czym dodaj 1.

(a) Wykaż, że następująca definicja jest równoważna. W przypadku n-bitowej liczby całkowitej X uzupełnienie do dwóch tworzy się, traktując X jako bezznakową liczbę całkowitą i obliczając $(2 \wedge n - X)$.

(b) Zademonstruj, że rysunek 9.5 może posłużyć do graficznego uzasadnienia twierdzenia (a) pokazując, w jaki sposób przeniesienie zgodne z kierunkiem zegara służy do realizowania odejmowania.

9.5. Oblicz następujące różnice, stosując uzupełnienie do dwóch

(a)	(b)	(c)	(d)
111000	11001100	111100001111	11000011
-110011	- 101110	-110011110011	-11101000

9.6. Czy właściwa jest następująca alternatywna definicja przepełnienia w arytmetyce uzupełnienia do dwóch?
Jeśli XOR (exclusive-OR) bitów przeniesienia: wchodzącego do najbardziej na lewo położonej kolumny i opuszczającego ją, jest równe 1, to występuje warunek przepełnienia. W przeciwnym razie nie występuje.

9.7. Porównaj rysunki 9.9 i 9.12. Dlaczego na tym ostatnim nie jest używany bit C?

9.8. Dane są $x = 0101$ i $y = 1010$ w notacji uzupełnienia do dwóch (tzn. $x = 5$, $y = -6$). Oblicz iloczyn $p = x \times y$ za pomocą algorytmu Bootha.

9.9. Udowodnij, że mnożenie dwóch n-cyfrowych liczb o podstawie B daje iloczyn o liczbie cyfr nie przekraczającej $2n$.

9.10. Zweryfikuj ważność algorytmu binarnego dzielenia bezznakowego z rys. 9.16, ukazując etapy dzielenia zilustrowane na rys. 9.15. Posłuż się prezentacją podobną do użytej na rys. 9.17.

9.11. Algorytm dzielenia liczb całkowitych w notacji uzupełnienia do dwóch opisany w podrozdz. 9.3 jest znany jako metoda przywracająca, ponieważ wartość rejestru A musi być przywrócona po nieudanym odejmowaniu. Nieco bardziej złożone podejście, znane jako nieprzywracające, zapobiega niekoniecznemu odejmowaniu i dodawaniu. Zaproponuj algorytm dla tego drugiego podejścia.

9.12. W arytmetyce liczb całkowitych iloraz J/K dwóch liczb całkowitych J i K jest mniejszy lub równy zwykłemu ilorazowi. Czy twierdzenie to jest prawdziwe?

9.13. Podziel –145 przez 13 w notacji binarnej uzupełnienia do dwóch, posługując się słowem 12-bitowym. Zastosuj algorytm opisany w p. 9.3.

9.14. Załóż, że wykładnik e jest ograniczony do zakresu $0 \leqslant e \leqslant X$, przy przesunięciu q, podstawie b i formacie o długości p cyfr.

(a) Jakie są największe i najmniejsze wartości dodatnie, które mogą być zapisane?

(b) Jakie są największe i najmniejsze wartości dodatnie, które mogą być zapisane jako znormalizowane liczby zmiennopozycyjne?

9.15. Wyraź następujące liczby w 32-bitowym formacie zmiennopozycyjnym IEEE:

(a) –5 (d) 384
(b) –6 (e) 1/16
(c) –1,5 (f) –1/32

9.16. Wyraź następujące liczby w 32-bitowym formacie zmiennopozycyjnym IBM, w którym jest stosowany 7-bitowy wykładnik i domyślna podstawa 16:

(a) 1,0 (e) –15,0
(b) 0,5 (f) $5{,}4 \times 10^{-79}$
(c) 1/64 (g) $7{,}2 \times 10^{75}$
(d) 0,0

9.17. Jaka powinna być wartość przesunięcia w przypadku:

(a) wykładnika podstawy 2 ($B = 2$) w polu 6-bitowym?
(b) wykładnika podstawy 8 ($B = 8$) w polu 7-bitowym?

9.18. Wykreśl oś liczbową podobną do tej z rys. 9.19b, dotyczącą formatów zmiennopozycyjnych z rys. 9.21b.

9.19. Rozważ format zmiennopozycyjny o 8-bitowym wykładniku przesuniętym i 23-bitowej mantysie. Podaj wzór bitowy następujących liczb w tym formacie:

(a) –720
(b) 0,645

9.20. Gdy ludzie mówią o niedokładności w arytmetyce zmiennopozycyjnej, opisują często błędy wynikające z kasowania, które ma miejsce podczas odejmowania prawie równych wielkości. Jednak gdy X i Y są w przybliżeniu równe, różnica $X - Y$ jest osiągana dokładnie, bez błędu. Co ci ludzie mają w rzeczywistości na myśli?

9.21. Jakakolwiek notacja zmiennopozycyjna stosowana w komputerze może reprezentować dokładnie tylko pewne liczby rzeczywiste; wszystkie inne muszą być aproksymowane. Jeśli A' jest przechowywaną wartością aproksymującą rzeczywistą wartość A, to błąd względny r jest wyrażany jako

$$r = \frac{A - A'}{A}$$

Przedstaw wielkość dziesiętną +0,4 w następującym formacie zmiennopozycyjnym: podstawa = 2, wykładnik przesunięty, 4-bitowy, mantysa 7-bitowa. Ile wynosi błąd względny?

9.22. Wartości numeryczne A i B są przechowywane w komputerze jako przybliżenia A' i B'. Zaniedbując jakiekolwiek inne błędy wynikające z obcięcia lub zaokrąglania, wykaż, że względny błąd iloczynu jest w przybliżeniu sumą błędów względnych obu czynników.

9.23. Jeśli $A = 1{,}427$, oblicz błąd względny przy A obciętym do 1,42 oraz przy A zaokrąglonym do 1,43.

9.24. Jeden z najpoważniejszych błędów w obliczeniach komputerowych występuje, gdy są odejmowane dwie prawie równe liczby. Przyjmij $A = 0{,}22288$ i $B = 0{,}22211$. Komputer obcina wszystkie wartości do czterech cyfr dziesiętnych. Wobec tego $A' = 0{,}2228$ oraz $B' = 0{,}2221$.

(a) Jakie są błędy względne A' i B'?
(b) Jaki jest błąd względny $C' = A' - B'$?

9.25. Pokaż, jak wykonuje się następujące operacje dodawania zmiennopozycyjnego (mantysy są skrócone do 4 cyfr dziesiętnych).

(a) $0{,}5566 \times 10^3 + 0{,}7777 \times 10^3$
(b) $0{,}3344 \times 10^2 + 0{,}8877 \times 10^{-1}$

9.26. Pokaż, jak wykonuje się następujące operacje odejmowania zmiennopozycyjnego (mantysy są skrócone do 4 cyfr dziesiętnych).

(a) $0{,}7744 \times 10^{-2} - 0{,}6666 \times 10^{-2}$
(b) $0{,}8844 \times 10^{-2} - 0{,}2233 \times 10^{0}$

9.27. Pokaż, jak wykonuje się następujące obliczenia zmiennopozycyjne (mantysy są skrócone do 4 cyfr dziesiętnych).

(a) $(0{,}2255 \times 10^{2}) \times (0{,}1234 \times 10^{1})$
(b) $(0{,}8833 \times 10^{3}) \div (0{,}5555 \times 10^{5})$

9.28. Wyraź następujące liczby ósemkowe w notacji szesnastkowej:

(a) 12
(b) 5655
(c) 2550276
(d) 76545336
(e) 3726755

9.29. Udowodnij, że każda liczba rzeczywista mająca skończoną reprezentację binarną (tzn. ze skończoną liczbą cyfr na prawo od przecinka binarnego) ma również skończoną reprezentację dziesiętną (tzn. skończoną liczbę cyfr na prawo od przecinka dziesiętnego).

9.16. Wyraź następujące liczby w 32-bitowym formacie [illegible], w którym jest stosowany 7-bitowy wykładnik i domyślna [illegible]

(b) 0,5 (f) [illegible]

(c) 1/64 (g) [illegible]

(d) 0,0

9.17. Jaka powinna być [illegible]

(a) wykładnika podstawy 2 ($B = 2$) w polu 8-bitowym?

(b) wykładnika podstawy 8 ($B = 8$) w polu 7-bitowym?

9.18. Wykreśl oś liczbową podobną do tej z rys. 9.19b, dotyczącą formatów zmiennopozycyjnych z rys. 9.21b.

9.19. Rozważ format zmiennopozycyjny o 8-bitowym wykładniku [illegible] i 23-bitowej mantysie. Podaj wzór bitowy następujących liczb w tym formacie:

[illegible]

9.20. Gdy ludzie mówią o niedokładności w arytmetyce zmiennopozycyjnej, opisują często błędy wynikające z kasowania, które ma miejsce podczas odejmowania prawie równych wielkości. Jednak gdy X i Y są w przybliżeniu równe, różnica $X - Y$ jest osiągana dokładnie, bez błędu. Co ci ludzie mają w rzeczywistości na myśli?

9.21. Jakakolwiek notacja zmiennopozycyjna stosowana w komputerze może reprezentować dokładnie tylko pewne liczby, natomiast wszystkie inne muszą być aproksymowane. Jeśli A' jest przechowywaną wartością aproksymującą rzeczywistą wartość A, to błąd względny r jest wyrażany jako

$$r = \frac{A - A'}{A}$$

Przedstaw wielkość dziesiętną +0,4 w następującym formacie zmiennopozycyjnym: podstawa = 2, wykładnik przesunięty, 4-bitowy, mantysa 7-bitowa. Ile wynosi błąd względny?

9.22. Wartości numeryczne A i B są przechowywane w komputerze jako przybliżenia A' i B'. Zaniedbując jakiekolwiek inne błędy wynikające z obcięcia lub zaokrąglenia, wykaż, że względny błąd iloczynu jest w przybliżeniu sumą błędów względnych obu czynników.

9.23. Jeśli $A = 1,427$, oblicz błąd względny przy A obciętym do 1,42 oraz przy A zaokrąglonym do 1,43.

9.24. Jeden z najpoważniejszych błędów w obliczeniach komputerowych występuje, gdy są odejmowane dwie prawie równe liczby. Przyjmij $A = 0,22288$ i $B = 0,22211$. Komputer obcina wszystkie wartości do czterech cyfr dziesiętnych. Wobec tego $A' = 0,2228$ oraz $B' = 0,2221$.

(a) Jakie są błędy względne A' i B'?

(b) Jaki jest błąd względny $C' = A' - B'$?

9.25. Pokaż, jak wykonuje się następujące operacje dodawania zmiennopozycyjnego (mantysy są skrócone do 4 cyfr dziesiętnych).

(a) $0,5566 \times 10^3 + 0,7777 \times 10^3$

(b) $0,3344 \times 10^2 + 0,8877 \times 10^{-1}$

9.26. Pokaż, jak wykonuje się następujące operacje odejmowania zmiennopozycyjnego (mantysy są skrócone do 4 cyfr dziesiętnych).

Rozdział 10

Listy rozkazów: właściwości i funkcje

10.1. Właściwości rozkazów maszynowych
10.2. Rodzaje argumentów
10.3. Rodzaje danych Pentium i PowerPC
10.4. Rodzaje operacji
10.5. Rodzaje operacji Pentium i PowerPC
10.6. Język asemblerowy
10.7. Polecana literatura
10.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Dodatek 10A. Stosy
Dodatek 10B. Struktury „cienko-", „grubo-" i „dwukońcowe".

Podstawowe spostrzeżenia

- Podstawowymi elementami rozkazu komputerowego są: kod operacji określający operację, która ma być wykonana; odwołania do argumentów źródłowych i miejsca docelowego, określające lokacje wejścia i wyjścia dla danej operacji oraz odwołanie do następnego rozkazu, które zwykle jest domyślne.
- Kody operacji określają operacje w ramach jednej spośród następujących kategorii ogólnych: operacje arytmetyczne i logiczne; przenoszenie danych między dwoma rejestrami, rejestrem a pamięcią lub między dwiema lokacjami w pamięci; sterowanie.
- Odwołania do argumentów określają rejestr lub lokację w pamięci danych będących argumentami. Dane te mogą być adresami, liczbami, znakami lub danymi logicznymi.
- Wspólną właściwością architektoniczną procesorów jest posługiwanie się stosem, który może być widzialny lub niewidzialny dla programisty. Stosy służą do zarządzania wywoływaniem i powrotami procedur oraz mogą stanowić alternatywną postać pamięci adresowania. Podstawowymi operacjami związanymi ze stosem są: PUSH (umieszczenie nowego elementu na wierzchołku stosu), POP (usunięcie elementu z wierzchołka stosu) oraz operacje na jednej lub dwóch wierzchołkowych lokacjach stosu. Stosy są zwykle implementowane tak, aby narastały od wyższych do niższych adresów.
- Procesory można podzielić na „grubokońcowe", „cienkokońcowe" i „dwukońcowe". Wielobajtowa wartość liczbowa zapisana tak, że najbardziej znaczący bajt znajduje się pod najniższym adresem liczbowym, jest zapisana w stylu grubokońcowym; jeśli zaś najbardziej znaczący bajt jest zapisany pod najwyższym adresem liczbowym, mamy do czynienia ze stylem cienkokońcowym. Procesory dwukońcowe mogą operować obydwoma stylami.

Użytkownik lub programista komputera nie widzi wielu obiektów opisanych w tej książce. Jeśli programista używa języka wysokiego poziomu, takiego jak Pascal lub Ada, niewiele dostrzega on z architektury używanej maszyny.

Jednym z obszarów, w których projektant komputera i jego programista mają wspólny obraz maszyny, jest lista rozkazów maszynowych. Z punktu widzenia projektanta, lista rozkazów maszynowych określa wymagania funkcjonalne w stosunku do procesora: implementacja procesora jest zadaniem, które w znacznej mierze polega na wdrożeniu listy rozkazów maszynowych. Z punktu widzenia użytkownika, tylko ten użytkownik, który wybiera programowanie w języku maszyny (w istocie w języku asemblerowym; patrz podrozdz. 9.4), uświadamia sobie strukturę rejestrów i pamięci, rodzaje danych bezpośrednio obsługiwanych przez maszynę oraz funkcjonowanie ALU.

Opis listy rozkazów maszynowych komputera oznacza duży krok naprzód w wyjaśnianiu działania procesora. Dlatego w tym i w następnym rozdziale skupiamy się na rozkazach maszynowych.

10.1. Właściwości rozkazów maszynowych

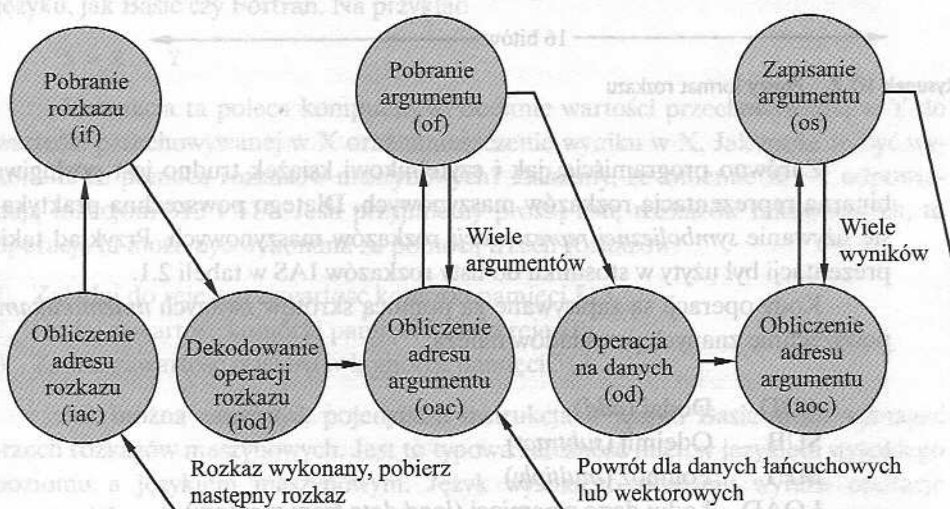
Działanie procesora jest określone przez rozkazy, które on wykonuje. Rozkazy te są nazywane *rozkazami maszynowymi* lub *rozkazami komputerowymi*. Zbiór różnych rozkazów, które może wykonywać procesor, jest określany jako *lista rozkazów*.

Elementy rozkazu maszynowego

Każdy rozkaz musi zawierać informacje wymagane przez procesor do jego wykonania. Na rysunku 10.1, będącym powtórzeniem rys. 3.6, są pokazane etapy wykonywania rozkazów. Przez implikację, etapy te definiują elementy rozkazu maszynowego. Elementami tymi są:

- **Kod operacji.** Określa operację, która ma być przeprowadzona (np. ADD, I/O). Operacja jest precyzowana za pomocą kodu binarnego – kodu operacji.
- **Odniesienie do argumentów źródłowych.** Operacja może obejmować jeden lub wiele argumentów źródłowych; są one danymi wejściowymi operacji.
- **Odniesienie do wyniku.** Operacja może prowadzić do powstania wyniku.
- **Odniesienie do następnego rozkazu.** Mówi ono procesorowi, skąd ma on pobrać następny rozkaz po zakończeniu wykonywania bieżącego rozkazu.

Następny rozkaz przewidziany do pobrania jest umieszczony w pamięci głównej lub, w przypadku systemu pamięci wirtualnej, albo w pamięci głównej, albo w pomocniczej (dyskowej). W większości przypadków następny rozkaz przewidziany do pobrania następuje bezpośrednio po bieżącym rozkazie. Nie występuje wówczas jawne odniesienie do następnego rozkazu. Gdy odniesienie to jest potrzebne, musi być dostarczony adres w pamięci głównej lub wirtualnej. Formę, w której jest dostarczany adres, opiszemy w rozdz. 11.



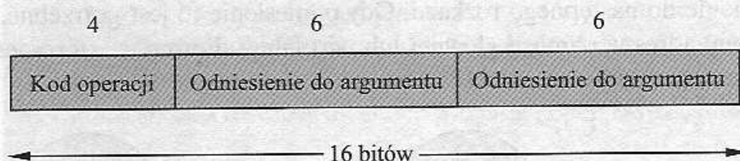
Rysunek 10.1. Graf stanów cyklu rozkazu

Argumenty źródłowe i wyniki mogą się znajdować w jednym z trzech obszarów. Te obszary to:

- **Pamięć główna lub wirtualna.** Podobnie jak w przypadku odniesienia do następnego rozkazu, musi być dostarczony adres pamięci.
- **Rejestr procesora.** Z rzadkimi wyjątkami procesor zawiera jeden lub wiele rejestrów, do których mogą się odnosić rozkazy maszynowe. Jeśli istnieje tylko jeden rejestr, odniesienie to może być domyślne. Jeśli istnieje więcej rejestrów niż jeden, to każdy rejestr ma przypisany unikatowy numer i rozkaz musi zawierać numer pożądanego rejestru.
- **Urządzenia wejścia-wyjścia.** Rozkaz musi określać moduł wejścia-wyjścia oraz urządzenie wejścia-wyjścia używane w operacji. Jeśli używane jest wejście-wyjście odwzorowane w pamięci, jest to jeszcze jeden adres pamięci głównej lub wirtualnej.

Reprezentacja rozkazu

Wewnątrz komputera każdy rozkaz jest reprezentowany za pomocą ciągu bitów. Rozkaz jest dzielony na pola odpowiadające elementom składowym rozkazu. Prosty przykład formatu rozkazu jest pokazany na rys. 10.2. Innym przykładem jest format rozkazu IAS pokazany na rys. 2.2. W większości list rozkazów używa się więcej niż jednego formatu. Rozkaz, podczas wykonywania, jest wczytywany do rejestru rozkazów (IR) w procesorze. Procesor musi być zdolny do pobrania danych z różnych pól rozkazu w celu wykonania wymaganej operacji.



Rysunek 10.2. Prosty format rozkazu

Zarówno programiście, jak i czytelnikowi książek trudno jest posługiwać się binarną reprezentacją rozkazów maszynowych. Dlatego powszechną praktyką stało się używanie *symbolicznej reprezentacji* rozkazów maszynowych. Przykład takiej reprezentacji był użyty w stosunku do listy rozkazów IAS w tabeli 2.1.

Kody operacji są zapisywane za pomocą skrótów zwanych *mnemonikami*. Do powszechnie znanych przykładów należą:

ADD	Dodaj (*add*)
SUB	Odejmij (*subtract*)
MPY	Pomnóż (*multiply*)
LOAD	Ładuj dane z pamięci (*load data from memory*)
STOR	Zapisz dane w pamięci (*store data to memory*).

Argumenty są również reprezentowane symbolicznie. Na przykład rozkaz

```
ADD R, Y
```

może oznaczać: dodaj wartość zawartą w lokacji danych Y do zawartości rejestru R. W tym przykładzie Y odnosi się do adresu komórki pamięci, R zaś oznacza określony rejestr. Zauważmy, że operacja jest przeprowadzana na zawartościach tych lokacji, a nie na adresach.

Jest więc możliwe napisanie programu w języku maszynowym w postaci symbolicznej. Każdy symboliczny kod operacji ma ustaloną reprezentację binarną, a programista określa lokalizację każdego symbolicznego argumentu. Programista może na przykład rozpocząć od listy definicji:

```
X = 513
Y = 514
```

i tak dalej. Prosty program akceptuje te symboliczne dane wejściowe, przekształca kody operacji i odniesienia do argumentów na postać binarną, po czym buduje binarne rozkazy maszynowe.

Osoby programujące w języku maszynowym są rzadkością. Większość dzisiejszych programów pisze się w języku wysokiego poziomu lub w języku asemblerowym, który będzie omówiony na końcu tego rozdziału. Jednak symboliczny język maszynowy pozostaje narzędziem użytecznym do opisania rozkazów maszynowych i do tego właśnie celu będziemy go używać.

Rodzaje rozkazów

Rozważmy rozkaz w języku wysokiego poziomu, który może być wyrażony w takim języku, jak Basic czy Fortran. Na przykład

```
X = X + Y
```

Instrukcja ta poleca komputerowi dodanie wartości przechowywanej w Y do wartości przechowywanej w X oraz umieszczenie wyniku w X. Jak może to być wykonane za pomocą rozkazów maszynowych? Załóżmy, że zmienne X i Y odpowiadają lokacjom 513 i 514. Jeśli przyjmiemy prostą listę rozkazów maszynowych, to operacja ta może być wykonana za pomocą trzech rozkazów:

1. Załaduj do rejestru zawartość komórki pamięci 513.
2. Dodaj zawartość komórki pamięci 514 do rejestru.
3. Zapisz zawartość rejestru w komórce pamięci 513.

Jak można zauważyć, pojedyncza instrukcja w języku Basic może wymagać trzech rozkazów maszynowych. Jest to typowa zależność między językiem wysokiego poziomu a językiem maszynowym. Język wysokiego poziomu wyraża operacje w zwartej formie algebraicznej, używając zmiennych. Język maszynowy wyraża operacje w postaci podstawowej, obejmując ruch danych do (lub z) rejestrów.

Posługując się tym prostym przykładem jako przewodnikiem, rozważmy rodzaje rozkazów, które muszą być przewidziane w rzeczywistym komputerze. Komputer powinien dysponować listą rozkazów, które umożliwiają użytkownikowi formułowanie dowolnych zadań dotyczących przetwarzania danych. Innym sposobem widzenia tego problemu jest rozważenie możliwości języka wysokiego poziomu. Dowolny program napisany w języku wysokiego poziomu musi być przetłumaczony na język maszynowy, aby mógł być wykonany. Wobec tego lista rozkazów maszynowych musi być wystarczająca do wyrażenia dowolnych instrukcji języka wysokiego poziomu. Mając to na uwadze, możemy podzielić rozkazy na następujące rodzaje:

- **Przetwarzanie danych.** Rozkazy arytmetyczne i logiczne.
- **Przechowywanie danych.** Rozkazy pamięciowe.
- **Ruch danych.** Rozkazy wejścia-wyjścia.
- **Sterowanie.** Rozkazy testowania i rozgałęzienia.

Rozkazy *arytmetyczne* zapewniają zdolność obliczeniową przetwarzania danych numerycznych. Rozkazy *logiczne* (Boole'a) operują na bitach słowa raczej jako na bitach niż na liczbach; wobec tego zapewniają one zdolność przetwarzania dowolnych innych rodzajów danych, jakie użytkownik może chcieć wykorzystywać. Operacje te są wykonywane głównie na danych znajdujących się w rejestrach procesora. Wobec tego muszą istnieć rozkazy *pamięciowe* służące do przenoszenia danych między pamięcią a rejestrami. Rozkazy *wejścia-wyjścia* są potrzebne do przenoszenia danych i programów do pamięci oraz wyników obliczeń do użytkownika. *Rozkazy testowania* są używane do sprawdzania wartości słów danych lub stanu obliczeń. *Rozkazy rozgałęzienia* są używane do przeskoczenia do innego zestawu rozkazów zależnie od podjętych decyzji.

Przeanalizujemy różne rodzaje rozkazów bardziej szczegółowo w dalszym ciągu rozdziału.

Liczba adresów

Jeden z tradycyjnych sposobów opisywania architektury procesora opiera się na liczbie adresów zawartych w każdym rozkazie. Wymiar ten stał się mniej znaczący wraz ze wzrostem złożoności CPU. Jednak przeanalizowanie tej własności jest użyteczne.

Jaka jest maksymalna liczba adresów, która może być potrzebna w rozkazie? Jest oczywiste, że operacje arytmetyczne i logiczne wymagają argumentów. Praktycznie wszystkie operacje arytmetyczne i logiczne są albo operacjami jednoargumentowymi, albo dwuargumentowymi. Wobec tego potrzebujemy maksymalnie dwóch adresów jako odniesienia do argumentów. Wynik operacji musi być przechowany, co sugeruje trzeci adres. Na zakończenie, po wykonaniu rozkazu, musi być pobrany następny rozkaz; potrzebny jest więc jego adres.

Z powyższego rozumowania wynika, że rozkaz powinien zawierać cztery odniesienia adresowe: dwa dotyczące argumentów, jedno wyniku i jedno następnego

rozkazu. W praktyce czteroadresowe rozkazy występują bardzo rzadko. Większość procesorów to jednostki jedno-, dwu- lub trzyadresowe, przy czym adres następnego rozkazu jest domyślny (uzyskiwany z licznika programu).

Na rysunku 10.3 znajduje się porównanie typowych rozkazów jedno-, dwu- i trzyadresowych, które mogą być użyte do obliczenia Y = (A – B)÷(C + D × E). Za pomocą trzech adresów każdy rozkaz ustala dwie lokacje argumentów i lokację wyniku. Ponieważ nie chcielibyśmy zmieniać lokacji argumentów, do przechowywania wyników pośrednich jest używana tymczasowa lokacja T. Zauważmy, że występują tu cztery rozkazy i że oryginalne wyrażenie ma pięć argumentów.

Rozkaz		Komentarz
SUB	Y, A, B	Y ← A – B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y – T

(a) rozkazy trójadresowe

Rozkaz	Komentarz
MOVE Y, A	Y ← A
SUB Y, B	Y ← Y – B
MOVE T, D	T ← D
MPY T, E	T ← T × E
ADD T, C	T ← T + C
DIV Y, T	Y ← Y÷T

(b) rozkazy dwuadresowe

Rozkaz	Komentarz
LOAD D	AC ← D
MPY E	AC ← AC × E
ADD C	AC ← AC + C
STOR Y	Y ← AC
LOAD A	AC ← A
SUB B	AC ← AC – B
DIV Y	AC ← AC – Y
STOR Y	Y ← AC

(c) rozkazy jednoadresowe

Rysunek 10.3. Programy do obliczenia Y = (A – B)÷(C + D × E)

Trzyadresowe formaty rozkazów nie są powszechne, ponieważ są stosunkowo długie w związku z koniecznością objęcia trzech odniesień adresowych. W przypadku rozkazów dwuadresowych oraz operacji binarnych jeden adres musi być używany podwójnie, zarówno jako adres argumentu, jak i wyniku. Wobec tego rozkaz SUB Y, B powoduje obliczenie Y – B oraz przechowanie wyniku w Y. Format dwuadresowy umożliwia zmniejszenie wymagań objętościowych, jednak wprowadza pewne niewygody. Aby zapobiec zmianie wartości argumentu, stosowany jest rozkaz MOVE, powodujący przeniesienie jednej z wartości do lokacji wyniku lub lokacji tymczasowej przed wykonaniem operacji. Nasz próbny program rozszerza się w ten sposób do sześciu rozkazów.

Jeszcze prostszy jest rozkaz jednoadresowy. Aby mógł on działać, drugi adres musi być domyślny. Było to powszechne we wcześniejszych maszynach, w których adres domyślny dotyczył rejestru procesora zwanego *akumulatorem* (AC). Akumulator zawiera jeden z argumentów i jest używany do przechowywania wyniku. Zadanie z naszego przykładu rozrasta się do ośmiu rozkazów.

W rzeczywistości jest możliwe, aby niektóre rozkazy funkcjonowały bez adresu. Rozkazy bezadresowe mogą być stosowane w przypadku szczególnej organizacji pamięci zwanej *stosem*. Stos jest zbiorem lokacji funkcjonującym na zasadzie ostatni na wejściu, pierwszy na wyjściu. Stos znajduje się w znanym położeniu i często przynajmniej jego dwa szczytowe elementy znajdują się w rejestrach procesora. Stosy są opisane w dodatku 10A. Używanie ich przeanalizujemy w dalszym ciągu tego rozdziału oraz w rozdz. 11.

W tabeli 10.1 znajduje się podsumowanie interpretacji rozkazów zawierających 0, 1, 2 lub 3 adresy. W każdym przypadku jest przyjęte, że adres następnego rozkazu jest domyślny oraz że jedna operacja wymaga dwóch argumentów i dostarcza jednego wyniku.

Tabela 10.1. Wykorzystanie adresów rozkazu (rozkazy nierozgałęziające)

Liczba adresów	Reprezentacja symboliczna	Interpretacja
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T − 1) OP T

AC = akumulator
T = wierzchołek stosu
A, B, C = lokacje (komórki) w pamięci lub w rejestrze
(T − 1) = Zawartość drugiego elementu stosu

Liczba adresów w rozkazie jest podstawową decyzją projektową. Mniej adresów w rozkazie oznacza prymitywniejsze rozkazy, wymagające prostszych procesorów. Oznacza to także krótsze rozkazy. Jednak programy zawierają wówczas więcej rozkazów, co na ogół powoduje wydłużenie czasów wykonywania oraz wydłużenie i skomplikowanie samych programów. Między rozkazami jedno- i wieloadresowymi występuje pewien próg. W przypadku rozkazów jednoadresowych programista ma na ogół dostęp tylko do jednego rejestru o ogólnym przeznaczeniu, tj. do akumulatora. W przypadku rozkazów wieloadresowych zwykle występuje wiele rejestrów o ogólnym przeznaczeniu. Dzięki temu niektóre operacje mogą być wykonywane na samych tylko rejestrach. Ponieważ dostęp do rejestrów jest szybszy niż do pamięci, przyspiesza to wykonywanie programów. Z powodu elastyczności i możliwości używania wielu rejestrów w większości współczesnych maszyn stosuje się mieszaninę rozkazów dwu- i trzyadresowych.

Kompromis projektowy związany z wyborem liczby adresów w rozkazie jest dodatkowo komplikowany przez inne czynniki. Istnieje problem, czy adres odnosi się do lokacji w pamięci, czy do rejestru. Ponieważ rejestrów jest mniej, odniesienie do rejestru wymaga mniejszej liczby bitów. Ponadto, jak zobaczymy w następnym rozdziale, maszyna może oferować różne tryby adresowania, a specyfikacja trybu zabiera jeden lub wiele bitów. W rezultacie w większości projektów procesorów przewidziano różne formaty rozkazów.

Projektowanie listy rozkazów

Jednym z najbardziej interesujących i najczęściej analizowanych aspektów projektowania komputera jest projektowanie listy rozkazów. Jest ono bardzo złożone, ponieważ wpływa na wiele aspektów systemu komputerowego. Lista rozkazów definiuje wiele funkcji realizowanych przez procesor i dlatego ma znaczący wpływ na implementację procesora. Lista rozkazów jest środkiem, za pomocą którego programista steruje procesorem. Dlatego też podczas projektowania listy rozkazów muszą być brane pod uwagę wymagania programisty.

Niespodzianką dla czytelnika może być to, że pewne najbardziej podstawowe problemy związane z projektowaniem list rozkazów nadal są przedmiotem debat. Istotnie, w ostatnich latach poziom niezgodności w odniesieniu do tych problemów wzrósł. Do najważniejszych spośród tych problemów projektowych należą:

- **Repertuar operacji.** Ile operacji, które operacje i jak złożone operacje powinny być przewidziane.
- **Rodzaje danych.** Różne rodzaje danych, na których dokonywane są operacje.
- **Format rozkazu.** Długość rozkazu (w bitach), liczba adresów, rozmiar różnych pól itd.
- **Rejestry.** Liczba rejestrów w procesorze, do których mogą się odnosić rozkazy, oraz ich zastosowanie.
- **Adresowanie.** Tryb lub tryby, w których są specyfikowane adresy argumentów.

Zagadnienia te są ze sobą ściśle powiązane i podczas projektowania listy rozkazów muszą być rozważane łącznie. Oczywiście, w tej książce muszą być one rozpatrzone po kolei, jednak spróbujemy pokazać ich wzajemne zależności.

Ze względu na wagę tego tematu, większość części trzeciej jest poświęcona projektowaniu listy rozkazów. Po tym podrozdziale mającym charakter przeglądu, dalej w tym rozdziale zajmiemy się analizą rodzaju danych i repertuarem operacji. W rozdziale 11 rozpatrzymy tryby adresowania (co obejmuje rozważenie rejestrów) oraz formaty rozkazów. W rozdziale 13 przedstawimy ekscytujące nowe rozwiązania znane jako komputery o zredukowanej liście rozkazów (RISC). Architektura RISC stawia pod znakiem zapytania wiele decyzji podjętych przy projektowaniu list rozkazów wielu współczesnych komputerów komercyjnych.

10.2. Rodzaje argumentów

Rozkazy maszynowe operują na danych. Najważniejsze, ogólne kategorie danych to:

- adresy,
- liczby,
- znaki,
- dane logiczne.

Analizując tryby adresowania w rozdz. 11, zobaczymy, że w rzeczywistości adresy są formą danych. W wielu przypadkach muszą być wykonywane pewne obliczenia na odniesieniach w rozkazach w celu wyznaczenia adresu w pamięci głównej lub wirtualnej. W tym kontekście adresy są uważane za liczby całkowite bez znaku.

Innymi powszechnymi rodzajami danych są liczby, znaki i dane logiczne. Każdy z nich zostanie krótko przeanalizowany w tym podrozdziale. Ponadto, w niektórych maszynach występują specjalizowane rodzaje danych lub struktur danych. Operatorzy maszyn mogą na przykład działać bezpośrednio na listach lub ciągach znaków.

Liczby

Wszystkie języki maszynowe obejmują dane numeryczne. Nawet w przypadku przetwarzania danych nienumerycznych potrzebne są liczby występujące w licznikach, określające szerokości pól itd. Ważną różnicą między liczbami używanymi w zwykłej matematyce a liczbami przechowywanymi w komputerze jest to, że te ostatnie są ograniczone. Jest to prawda w podwójnym sensie. Po pierwsze, istnieje granica wielkości liczb reprezentowalnych w komputerze, a po drugie, w przypadku liczb zmiennopozycyjnych, granica ich dokładności. Programista powinien więc znać konsekwencje zaokrąglania, przepełnienia i niedomiaru.

W komputerach są powszechne trzy rodzaje danych numerycznych:

- całkowite lub stałopozycyjne,
- zmiennopozycyjne,
- dziesiętne.

Dwa pierwsze rodzaje przeanalizowaliśmy dość szczegółowo w rozdz. 9. Pozostaje więc powiedzieć kilka słów o liczbach dziesiętnych.

Chociaż wszystkie wewnętrzne operacje komputera są z natury binarne, użytkownicy systemu operują liczbami dziesiętnymi. Istnieje więc konieczność konwersji liczb dziesiętnych na binarne na wejściu oraz binarnych na dziesiętne na wyjściu. W przypadku zastosowań, w których występuje wiele operacji wejścia-wyjścia oraz raczej mało stosunkowo prostych obliczeń, wygodniej jest przechowywać i przetwarzać liczby w postaci dziesiętnej. Najbardziej powszechną reprezentacją służącą do tego celu są upakowane liczby dziesiętne.

W przypadku upakowanych liczb dziesiętnych każda cyfra dziesiętna jest reprezentowana w kodzie 4-bitowym, tworzonym w oczywisty sposób. Wobec tego 0=0000, 1=0001, ... , 8=1000 i 9=1001. Zauważmy, że jest to kod raczej nieefektywny, ponieważ używa tylko 10 spośród możliwych 16 wartości 4-bitowych. W celu utworzenia liczb 4-bitowe kody są zestawiane szeregowo, zwykle w wielokrotnościach 8 bitów. Kodem 246 jest więc 0000001001000110. Kod ten jest oczywiście mniej zwarty niż prosta reprezentacja binarna, jednak pozwala uniknąć konwersji. Liczby ujemne mogą być reprezentowane przez dołączenie 4-bitowej cyfry znaku albo z lewej, albo z prawej strony ciągu upakowanych cyfr dziesiętnych. Na przykład kod 1111 może reprezentować znak minus.

W wielu maszynach przewidziano rozkazy arytmetyczne do wykonywania operacji bezpośrednio na upakowanych liczbach dziesiętnych. Algorytmy te są całkiem podobne do opisanych w podrozdz. 9.3, jednak muszą uwzględniać operację przeniesienia dziesiętnego.

Znaki

Powszechną formą danych jest tekst lub ciągi znaków. Chociaż dane tekstowe są najwygodniejsze dla ludzi, nie mogą one być w postaci znaków łatwo przechowywane i transmitowane przez systemy przetwarzania i komunikacji. Systemy te są zaprojektowane dla danych binarnych. W związku z tym opracowano wiele kodów, za pomocą których znaki są reprezentowane w postaci ciągów bitów. Być może najwcześniejszym, powszechnie znanym przykładem jest kod Morse'a. Dzisiaj najczęściej stosowanym kodem znaków jest International Reference Alphabet (Międzynarodowy Alfabet Wzorcowy, IRA), w Stanach Zjednoczonych znany jako American Standard Code for Information Exchange (Standardowy Amerykański Kod Wymiany Informacji, ASCII, patrz tab. 7.1). IRA jest również używany szeroko poza Stanami Zjednoczonymi. Każdy znak w tym kodzie jest reprezentowany przez unikatowy wzór 7-bitowy; może więc być reprezentowanych 128 różnych znaków. Jest to liczba większa niż wymagana do reprezentowania znaków drukowanych, dlatego część wzorów reprezentuje znaki *sterowania*. Niektóre z tych znaków sterowania są stosowane do sterowania drukowaniem znaków na stronie. Inne wiążą się z procedurami komunikacyjnymi. Znaki zakodowane za pomocą IRA są prawie zawsze przechowywane i transmitowane przy użyciu 8 bitów na znak. Ósmy bit może być ustalony jako 0 lub użyty jako bit parzystości do wykrywania błędów. W tym ostatnim przypadku bit jest ustalany w ten sposób, że całkowita liczba binarnych jedynek w każdym oktecie jest zawsze nieparzysta (parzystość nieparzysta) lub zawsze parzysta (parzystość parzysta).

Zauważmy, że dla wzoru bitowego IRA 011XXXX, cyfry od 0 do 9 są reprezentowane przez ich binarne równoważniki, 0000 do 1001, zlokalizowane w prawej części wzoru. Jest to ten sam kod, co stosowany w przypadku upakowanych liczb dziesiętnych. Ułatwia to konwersję między 7-bitowym kodem IRA a 4-bitową upakowaną reprezentacją dziesiętną.

Innym kodem używanym do kodowania znaków jest EBCDIC (*Extended Binary Coded Decimal Interchange Code*). Kod EBCDIC jest stosowany w maszynach IBM S/390. Jest to kod 8-bitowy. Podobnie jak IRA, EBCDIC jest kompatybilny z upakowaną reprezentacją dziesiętną. W przypadku EBCDIC kody od 11110000 do 1111001 reprezentują cyfry od 0 do 9.

Dane logiczne

Zwykle każde słowo lub inna jednostka adresowalna (bajt, półsłowo itd.) jest traktowane jako jednostka danych. Jest jednak czasem użyteczne rozpatrywanie jednostki n-bitowej jako składającej się z n jednobitowych pozycji, z których każda ma wartość 1 lub 0. Dane widziane w ten sposób są traktowane jako dane *logiczne*.

Ten zorientowany bitowo obraz ma dwie zalety. Po pierwsze, czasem może być potrzebne przechowywanie tablicy boolowskich lub binarnych danych, z których każda może przybierać tylko wartości 1 (prawdziwa) i 0 (fałszywa). Dane w postaci logicznej mogą być bardziej efektywnie przechowywane w pamięci. Po drugie, występują sytuacje, w których chcemy manipulować bitami jednostki danych. Jeśli na przykład w oprogramowaniu są wdrażane operacje zmiennopozycyjne, to w pewnych operacjach zachodzi potrzeba przesuwania znaczących bitów. Inny przykład: w celu konwersji kodu IRA na upakowany kod dziesiętny musimy wyciągnąć 4 prawe bity z każdego bajta.

Zauważmy, że w powyższych przykładach te same dane są traktowane czasem jako logiczne, a czasem jako numeryczne lub tekstowe. „Rodzaj" jednostki danych jest określany przez wykonywaną na nich operację. Nie jest to normalne w przypadku języków wysokiego poziomu, ma to jednak miejsce prawie zawsze w przypadku języka maszynowego.

10.3. Rodzaje danych Pentium i PowerPC

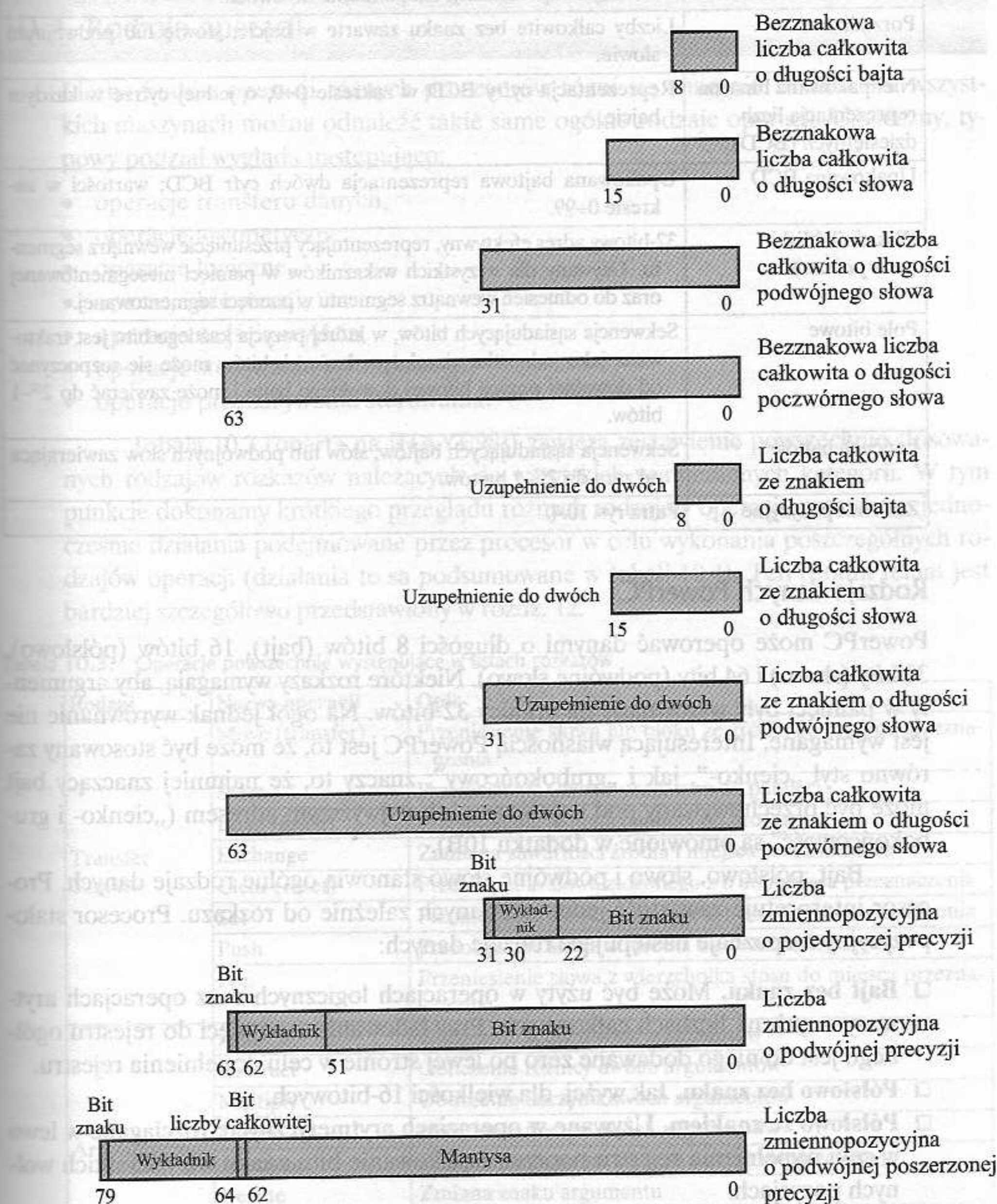
Rodzaje danych Pentium

Pentium może operować danymi o długości 8 bitów (bajt), 16 bitów (słowo), 32 bity (podwójne słowo) i 64 bity (poczwórne słowo). Aby osiągnąć maksymalną elastyczność struktur danych oraz efektywne wykorzystanie pamięci, słowa nie muszą być wyrównywane pod adresami numerowanymi parzyście, podwójne słowa nie muszą być wyrównywane pod adresami podzielnymi przez cztery, a poczwórne słowa nie muszą być wyrównywane pod adresami podzielnymi przez 8. Jeśli jednak dostęp do danych następuje przez magistralę 32-bitową, przesyłanie danych następuje w podwójnych słowach jako jednostkach, począwszy od adresów podzielnych przez 4. Procesor zamienia zapotrzebowanie odniesione do niewyrównanych wartości na sekwencję zapotrzebowań dostosowanych do transferu magistralowego. Podobnie jak w przypadku wszystkich procesorów Intel 80×86, Pentium używa stylu określanego jako „cienkokońcowy"* (*little-endian*); oznacza to, że najmniej znaczący bajt jest przechowywany pod najniższym adresem (patrz dodatek 10B zawierający omówienie „cienko-" i „grubokońcowości").

Bajt, słowo, słowo podwójne i słowo poczwórne są określane jako ogólne rodzaje danych. Ponadto Pentium przyjmuje obszerny zestaw specyficznych rodzajów danych, które są rozpoznawane i uruchamiane za pomocą poszczególnych rozkazów. Są one wymienione w tabeli 10.2.

Na rysunku 10.4 zostały przedstawione rodzaje danych numerycznych Pentium. Liczby całkowite ze znakiem mają postać uzupełnienia do dwóch i mogą mieć długość 16, 32 lub 64 bitów. Zmiennopozycyjny rodzaj danych odnosi się w rzeczywistości do pewnego zbioru rodzajów, które są używane przez jednostkę zmiennopozycyjną i wykorzystywane przez rozkazy zmiennopozycyjne. Trzy reprezentacje zmiennopozycyjne odpowiadają normie IEEE 754.

* Zobacz przypis na s. 422 w dodatku 10B (*przyp. red.*).



Rysunek 10.4. Formaty danych numerycznych jednostki zmiennopozycyjnej Pentium

Tabela 10.2. Rodzaje danych procesora Pentium

Rodzaj danych	Opis
Ogólne	Lokacje o dowolnej zawartości binarnej obejmujące bajt, słowo (16 bitów), podwójne słowo (32 bity) lub poczwórne słowo (64 bity).
Całkowite	Wartości binarne ze znakiem, zawarte w bajcie, słowie lub podwójnym słowie, w reprezentacji uzupełnienia do dwóch.
Porządkowe	Liczby całkowite bez znaku zawarte w bajcie, słowie lub podwójnym słowie.
Nieupakowana binarna reprezentacja liczb dziesiętnych (BCD)	Reprezentacja cyfry BCD w zakresie 0÷9, o jednej cyfrze w każdym bajcie.
Upakowana BCD	Upakowana bajtowa reprezentacja dwóch cyfr BCD; wartości w zakresie 0÷99.
Wskaźnik bliski (*near pointer*)	32-bitowy adres efektywny, reprezentujący przesunięcie wewnątrz segmentu. Używany dla wszystkich wskaźników w pamięci niesegmentowanej oraz do odniesień wewnątrz segmentu w pamięci segmentowanej.
Pole bitowe	Sekwencja sąsiadujących bitów, w której pozycja każdego bitu jest traktowana jako jednostka niezależna. Łańcuch bitów może się rozpoczynać od dowolnej pozycji bitowej dowolnego bajta i może zawierać do $2^{32}-1$ bitów.
Łańcuch bajtów	Sekwencja sąsiadujących bajtów, słów lub podwójnych słów zawierająca od zera do $2^{32}-1$ bajtów.
Zmiennopozycyjne	Patrz rys. 10.4.

Rodzaje danych PowerPC

PowerPC może operować danymi o długości 8 bitów (bajt), 16 bitów (półsłowo), 32 bity (słowo) i 64 bity (podwójne słowo). Niektóre rozkazy wymagają, aby argumenty w pamięci były wyrównane na granicy 32 bitów. Na ogół jednak wyrównanie nie jest wymagane. Interesującą własnością PowerPC jest to, że może być stosowany zarówno styl „cienko-", jak i „grubokońcowy"; znaczy to, że najmniej znaczący bajt może być przechowywany pod najniższym lub najwyższym adresem („cienko- i grubokońcowość" są omówione w dodatku 10B).

Bajt, półsłowo, słowo i podwójne słowo stanowią ogólne rodzaje danych. Procesor interpretuje zawartość jednostki danych zależnie od rozkazu. Procesor stałopozycyjny rozpoznaje następujące rodzaje danych:

- **Bajt bez znaku.** Może być użyty w operacjach logicznych oraz operacjach arytmetycznych na liczbach całkowitych. Przy ładowaniu z pamięci do rejestru ogólnego jest do niego dodawane zero po lewej stronie w celu dopełnienia rejestru.
- **Półsłowo bez znaku.** Jak wyżej, dla wielkości 16-bitowych.
- **Półsłowo ze znakiem.** Używane w operacjach arytmetycznych; rozciągane w lewo w celu wypełnienia rejestru poprzez replikowanie bitu znaku we wszystkich wolnych pozycjach.
- **Słowo bez znaku.** Używane w operacjach logicznych i jako wskaźnik adresu.
- **Słowo ze znakiem.** Używane w operacjach arytmetycznych.

- **Podwójne słowo bez znaku.** Używane jako wskaźnik adresu.
- **Łańcuch bajtów.** Od 0 do 128 bajtów długości.

Ponadto PowerPC operuje na danych zmiennopozycyjnych o pojedynczej i podwójnej dokładności, zdefiniowanych w IEEE 754.

10.4. Rodzaje operacji

Liczba kodów operacji różnych procesorów różni się znacznie. Jednak we wszystkich maszynach można odnaleźć takie same ogólne rodzaje operacji. Użyteczny, typowy podział wygląda następująco:

- operacje transferu danych,
- operacje arytmetyczne,
- operacje logiczne,
- operacje konwersji,
- operacje wejścia-wyjścia,
- operacje sterowania systemowego,
- operacje przekazywania sterowania.

Tabela 10.3 (oparta na [HAYE98]) zawiera zestawienie powszechnie stosowanych rodzajów rozkazów należących do wszystkich wymienionych kategorii. W tym punkcie dokonamy krótkiego przeglądu różnych rodzajów operacji, omawiając jednocześnie działania podejmowane przez procesor w celu wykonania poszczególnych rodzajów operacji (działania te są podsumowane w tabeli 10.4). Ten ostatni temat jest bardziej szczegółowo przedstawiony w rozdz. 12.

Tabela 10.3. Operacje powszechnie występujące w listach rozkazów

Rodzaj	Nazwa operacji	Opis
Transfer danych	Move (transfer)	Przeniesienie słowa lub bloku ze źródła do miejsca przeznaczenia
	Store	Przeniesienie słowa z procesora do pamięci
	Load (fetch)	Przeniesienie słowa z pamięci do procesora
	Exchange	Zamiana zawartości źródła i miejsca przeznaczenia
	Clear (reset)	Przeniesienie słowa złożonego z 0 do miejsca przeznaczenia
	Set	Przeniesienie słowa złożonego z 1 do miejsca przeznaczenia
	Push	Przeniesienie słowa ze źródła na wierzchołek stosu
	Pop	Przeniesienie słowa z wierzchołka stosu do miejsca przeznaczenia
Arytmetyczne	Add	Obliczenie sumy dwóch argumentów
	Subtract	Obliczenie różnicy dwóch argumentów
	Multiply	Obliczenie iloczynu dwóch argumentów
	Divide	Obliczenie ilorazu dwóch argumentów
	Absolute	Zamiana argumentu na jego wartość bezwzględną
	Negate	Zmiana znaku argumentu
	Increment	Dodanie 1 do argumentu
	Decrement	Odjęcie 1 od argumentu

Tabela 10.3. Operacje powszechnie występujące w listach rozkazów (cd.)

Rodzaj	Nazwa operacji	Opis
Logiczne	AND OR NOT (complement) Exclusive-OR	Wykonanie określonej operacji logicznej na poziomie bitowym
	Test	Zbadanie określonego warunku; jako wynik ustawienie znacznika (znaczników) stanu
	Compare	Wykonanie logicznego lub arytmetycznego porównania dwóch lub wielu argumentów; na podstawie wyniku ustawienie znacznika (znaczników) stanu
	Set control variables	Klasa rozkazów ustalających elementy sterowania dla celów ochrony, przetwarzania przerwań, sterowania zegarowego
	Shift	Przesunięcie argumentu w prawo lub w lewo z wprowadzeniem stałej na końcu
	Rotate	Przesunięcie argumentu w prawo lub w lewo z łączeniem końca z początkiem
Przeniesienie sterowania	Jump (branch)	Przeniesienie bezwarunkowe; ładuje określony adres do PC
	Jump Conditional	Zbadanie określonego warunku; albo załadowanie określonego adresu do PC, albo nie robienie niczego, zależnie od warunku
	Jump to Subroutine	Umieszczenie informacji kontrolnej bieżącego programu w znanym miejscu; skok do określonego adresu
	Return	Zamiana zawartości PC i innych rejestrów na dane przechowywane w znanym miejscu
	Execute	Pobranie argumentu z określonego miejsca i wykonanie go jako rozkazu; nie modyfikuje PC
	Skip	Inkrementowanie PC w celu pominięcia następnego rozkazu
	Skip conditional	Zbadanie określonego warunku; albo pominięcie następnego rozkazu, albo nie robienie niczego, zależnie od warunku
	Halt	Zatrzymanie wykonywania programu
	Wait (hold)	Zatrzymanie wykonywania programu; powtarzalne badanie określonego warunku; wykonywanie wznawiane po spełnieniu warunku
	No operation	Nie jest wykonywana żadna operacja, kontynuowane jest wykonywanie programu
Wejście-wyjście	Input (read)	Przeniesienie danych z określonego portu lub przyrządu we-wy do miejsca przeznaczenia, np. do pamięci głównej lub rejestru procesora
	Output (write)	Przeniesienie danych z określonego źródła do portu lub przyrządu we-wy
	Start I/O	Przeniesienie rozkazów do procesora we-wy w celu inicjowania operacji we-wy
	Test I/O	Przeniesienie informacji o stanie z systemu we-wy do określonego miejsca przeznaczenia
Konwersja	Translate	Przetłumaczenie danych w sekcji pamięci na podstawie tabeli odpowiedniości
	Convert	Przekształcenie zawartości słowa z jednej postaci na drugą (np. z upakowanej dziesiętnej na binarną)

Tabela 10.4. Czynności wykonywane przez procesor w przypadku różnych rodzajów operacji

Transfer danych	Przeniesienie danych z jednego miejsca do drugiego Jeśli jest zaangażowana pamięć: określenie adresu pamięci, przekształcenie adresu wirtualnego na rzeczywisty, sprawdzenie pamięci podręcznej, zainicjowanie zapisu/odczytu pamięci
Arytmetyczne	Mogą obejmować transfer danych przed i (lub) po wykonaniu działania. Wykonanie działania w ALU, Ustawienie kodu warunkowego i znacznika stanu
Logiczne	Jak arytmetyczne
Konwersja	Podobnie do arytmetycznych i logicznych. Mogą obejmować specjalne funkcje logiczne w celu dokonania konwersji
Transfer sterowania	Aktualizowanie licznika programu. W przypadku wywołania/powrotu z podprogramu – zarządzanie przenoszeniem i powiązaniem parametrów
Wejścia-wyjścia	Wydanie rozkazu modułowi wejścia-wyjścia. Jeśli wejście-wyjście jest odwzorowane w pamięci, określenie adresu odwzorowanego w pamięci

Transfer danych

Najbardziej podstawowym rodzajem rozkazów maszynowych jest rozkaz przesyłania (transferu) danych. Rozkaz taki musi precyzować kilka rzeczy. Po pierwsze, musi być ustalone położenie argumentów źródłowych i wyników. Możliwe położenia obejmują pamięć, rejestr lub wierzchołek stosu. Po drugie, musi być określona długość danych przeznaczonych do przesłania. Po trzecie, tak jak w przypadku wszystkich rozkazów z argumentami, musi być określony tryb adresowania każdego argumentu. To ostatnie zagadnienie jest omówione w rozdz. 11.

Wybór rozkazów transferu danych, które mają być włączone do listy rozkazów, stanowi egzemplifikację wyborów, jakich musi dokonywać projektant. Na przykład ogólne położenie (pamięć lub rejestr) argumentu może być wskazane albo w specyfikacji kodu operacji, albo samego argumentu. W tabeli 10.5 są podane przykłady najczęściej stosowanych rozkazów transferu danych IBM S/390. Zauważmy, że występują tutaj różne warianty ilości danych przeznaczonych do przesłania (8, 16, 32 i 64 bity). Istnieją też różne rozkazy dotyczące transferów z rejestru do rejestru, z rejestru do pamięci i z pamięci do rejestru. Dla kontrastu, w maszynie VAX występuje rozkaz przeniesienia (MOV) z wariantami dotyczącymi różnych ilości danych, jednak określa on, czy argument jest w rejestrze, czy w pamięci. Podejście stosowane w komputerze VAX jest nieco łatwiejsze dla programisty, który operuje mniejszą ilością mnemoników. Jest ono jednak mniej zwarte niż rozwiązanie rodem z IBM S/390, ponieważ lokalizacja (rejestr czy pamięć) każdego argumentu musi być wyspecyfikowana w rozkazie. Wrócimy do tego rozróżnienia przy dyskusji nad formatami rozkazów w następnym rozdziale.

Tabela 10.5. Przykłady operacji transferu danych w IBM S/390

Mnemonik operacji	Nazwa	Liczba przenoszonych bitów	Opis
L	Load	32	Przeniesienie z pamięci do rejestru
LH	Load halfword	16	Przeniesienie z pamięci do rejestru
LR	Load	32	Przeniesienie z rejestru do rejestru
LER	Load (short)	32	Przeniesienie z rejestru zmiennopozycyjnego do rejestru zmiennopozycyjnego
LE	Load (short)	32	Przeniesienie z pamięci do rejestru zmiennopozycyjnego
LDR	Load (long)	64	Przeniesienie z rejestru zmiennopozycyjnego do rejestru zmiennopozycyjnego
LD	Load (long)	64	Przeniesienie z pamięci do rejestru zmiennopozycyjnego
ST	Store	32	Przeniesienie z rejestru do pamięci
STH	Store halfword	16	Przeniesienie z rejestru do pamięci
STC	Store character	8	Przeniesienie z rejestru do pamięci
STE	Store (short)	32	Przeniesienie z rejestru zmiennopozycyjnego do pamięci
STD	Store (long)	64	Przeniesienie z rejestru zmiennopozycyjnego do pamięci

W kategoriach działań procesora, operacje transferu danych są być może rodzajem najprostszym. Jeśli zarówno źródłem, jak i miejscem przeznaczenia są rejestry, to procesor po prostu powoduje przeniesienie danych z jednego rejestru do drugiego; jest to operacja wewnętrzna procesora. Jeśli jeden lub oba argumenty znajdują się w pamięci, to procesor musi wykonać niektóre lub wszystkie z następujących działań:

1. Obliczanie adresu pamięci na podstawie określonego trybu adresowania (omówionego w rozdz. 11).
2. Jeśli adres dotyczy pamięci wirtualnej, przekształcenie adresu z wirtualnego na rzeczywisty.
3. Sprawdzenie, czy adresowana jednostka znajduje się w pamięci podręcznej.
4. Jeśli nie, wydanie rozkazu modułowi pamięci.

Operacje arytmetyczne

Większość maszyn realizuje podstawowe operacje arytmetyczne dodawania, odejmowania, mnożenia i dzielenia. Zawsze są one przewidziane dla liczb całkowitych (stałopozycyjnych) ze znakiem. Często są również przewidziane dla liczb zmiennopozycyjnych i upakowanych dziesiętnych.

Inne możliwe operacje obejmują różne rozkazy jednoargumentowe. Są to na przykład:

- **Wartość bezwzględna.** Określenie wartości bezwzględnej argumentu.
- **Negacja.** Zanegowanie argumentu.
- **Inkrementacja.** Przyrost argumentu o 1.
- **Dekrementacja.** Zmniejszenie argumentu o 1.

Wykonywanie rozkazu arytmetycznego może obejmować operacje przesłania danych w celu umieszczenia ich na wejściu ALU oraz w celu wyprowadzenia ich z wyjścia ALU. Na rysunku 3.5 został pokazany ruch danych objęty operacjami przesyłania danych oraz operacjami arytmetycznymi. Ponadto – oczywiście – część procesora w postaci ALU wykonuje żądaną operację.

Operacje logiczne

Większość maszyn umożliwia różne operacje manipulowania (*twiddling*) pojedynczymi bitami słowa lub innych adresowalnych jednostek. Są one oparte na operacjach Boole'a (patrz dodatek A do książki).

Niektóre z podstawowych operacji logicznych, które mogą być dokonywane na danych boolowskich lub binarnych, są podane w tabeli 10.6. Operacja NOT (NIE) odwraca bit. Operacje AND (I), OR (LUB) i Exclusive OR – XOR (LUB wykluczające) są najczęściej używanymi funkcjami logicznymi z dwoma argumentami. Operacja EQUAL (RÓWNY) jest użytecznym testem binarnym.

Tabela 10.6. Podstawowe operacje logiczne

P	Q	NOT P	NOT Q	P AND Q	P OR Q	P XOR Q	P=Q
0	0	1	1	0	0	0	1
0	1	1	0	0	1	1	0
1	0	0	1	0	1	1	0
1	1	0	0	1	1	0	1

Te operacje logiczne mogą być stosowane do bitów należących do *n*-bitowych logicznych jednostek danych. Jeśli więc dwa rejestry zawierają następujące dane:

(R1)=10100101

(R2)=00001111

to

(R1) AND (R2)=00000101

gdzie zapis (X) oznacza zawartość lokacji (X). Wobec tego operacja AND może być użyta jako *maska*, umożliwiająca wybranie pewnych bitów w słowie i zerowanie pozostałych bitów. Weźmy inny przykład; jeśli rejestry zawierają:

(R1) = 10100101

(R2) = 11111111

to

(R1) XOR (R2) = 01011010

Gdy jedno ze słów składa się z samych jedynek, to operacja XOR odwraca wszystkie bity w drugim słowie (uzupełnienie jedynki).

Poza bitowymi operacjami logicznymi większość maszyn umożliwia wiele funkcji przesuwania i rotacji. Najbardziej podstawowe operacje są przedstawione na rys. 10.5. Operacja **przesunięcia logicznego** powoduje przesunięcie bitów słowa w lewo lub w prawo. Z jednej strony jeden bit jest tracony, z drugiej wprowadzane jest 0. Przesunięcia logiczne są używane głównie do izolowania pola wewnątrz słowa. Zera wprowadzane do słowa zastępują niepożądaną informację, która jest wyprowadzana.

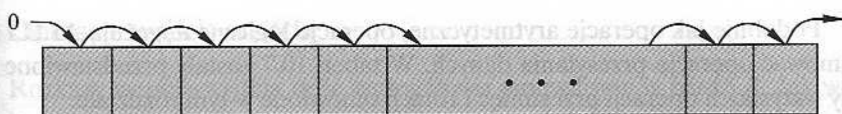
Załóżmy na przykład, że chcemy transmitować znaki do urządzenia wejścia-wyjścia metodą znak po znaku. Jeśli każde słowo pamięci ma długość 16 bitów i zawiera 2 znaki, to musimy „odpakować" znaki, zanim zostaną one wysłane. W celu wysłania 2 znaków zawartych w słowie są potrzebne następujące działania:

1. Załadowanie słowa do rejestru.
2. Wykonanie operacji AND z wartością 1111111100000000. Maskuje to znak po prawej.
3. Przesunięcie w prawo osiem razy. Powoduje to przesunięcie pozostałego znaku do prawej połowy rejestru.
4. Wykonanie operacji wejścia-wyjścia. Moduł wejścia-wyjścia odczyta 8 najmłodszych bitów z magistrali danych.

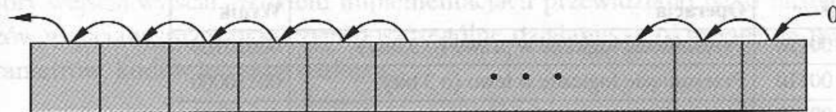
W wyniku wykonania tych czynności zostanie wysłany lewy znak. W celu wysłania prawego znaku należy:

1. Ponownie załadować słowo do rejestru.
2. Wykonać AND z 0000000011111111.
3. Wykonać operację wejścia-wyjścia.

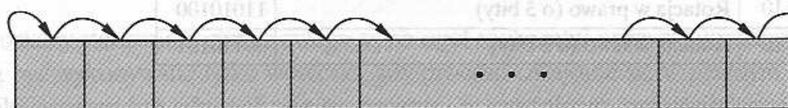
Przy wykonaniu operacji **przesunięcia arytmetycznego** dane są traktowane jako liczby całkowite ze znakami; bit znaku nie jest przesuwany. W przypadku przesunięcia arytmetycznego w prawo, bit znaku jest zwykle replikowany w najbliższej pozycji bitowej na prawo. Operacje te mogą przyspieszać pewne operacje arytmetyczne. W odniesieniu do liczb w notacji uzupełnienia do dwóch arytmetyczne przesunięcie w prawo odpowiada dzieleniu przez 2 z odcięciem w przypadku liczb nieparzystych. Zarówno arytmetyczne, jak i logiczne przesunięcie w lewo odpowiada mnożeniu przez 2 pod warunkiem, że nie występuje przepełnienie. Jeśli przepełnienie nie występuje, operacje arytmetycznego i logicznego przesunięcia w lewo prowadzą do odmiennych wyników, jednak arytmetyczne przesunięcie w lewo zachowuje znak liczby. Ze względu na możliwość przepełnienia, w przypadku wielu procesorów rozkaz ten nie występuje; do takich procesorów należą PowerPC i Itanium. W pozostałych



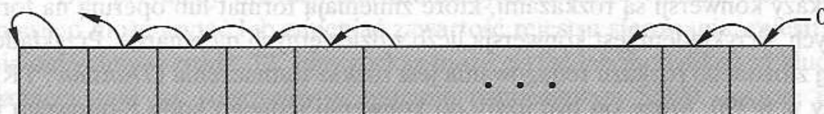
(a) Logiczne przesunięcie w prawo



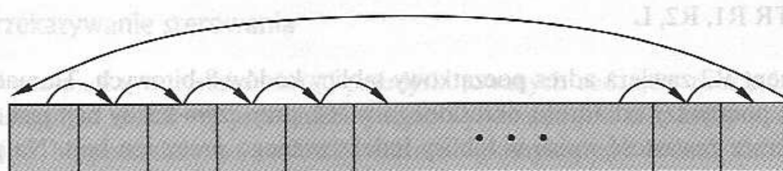
(b) Logiczne przesunięcie w lewo



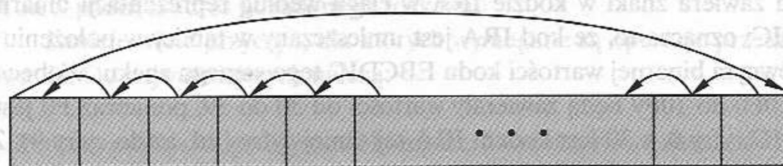
(c) Arytmetyczne przesunięcie w prawo



(d) Arytmetyczne przesunięcie w lewo



(e) Rotacja w prawo



(f) Rotacja w lewo

Rysunek 10.5. Operacje przesunięcia i rotacji

– takich jak IBM S/390 – rozkaz taki istnieje. Co zadziwiające, w architekturze Pentium zostało uwzględnione arytmetyczne przesunięcie w lewo, jednak jest ono zdefiniowane jako identyczne z logicznym.

Operacje **rotacji** lub **przesunięcia cyklicznego** zachowują wszystkie bity, na których są przeprowadzane. Jednym z możliwych zastosowań rotacji jest sukcesywne doprowadzanie każdego bitu do skrajnej lewej pozycji, gdzie może on być zidentyfikowany przez sprawdzenie znaku danych (traktowanych jako liczba).

Podobnie jak operacje arytmetyczne, operacje logiczne angażują ALU i mogą obejmować operacje przesyłania danych. W tabeli 10.7 zostały przedstawione przykłady wszystkich operacji przesunięć i rotacji omówione w tym rozdziale.

Tabela 10.7. Przykłady operacji przesuwania i rotacji

Wejście	Operacja	Wynik
10100110	Przesunięcie logiczne w prawo (o 3 bity)	00010100
10100110	Przesunięcie logiczne w lewo (o 3 bity)	00110000
10100110	Przesunięcie arytmetyczne w prawo (o 3 bity)	11110100
10100110	Przesunięcie arytmetyczne w lewo (o 3 bity)	10110000
10100110	Rotacja w prawo (o 3 bity)	11010100
10100110	Rotacja w lewo (o 3 bity)	00110101

Konwersja

Rozkazy konwersji są rozkazami, które zmieniają format lub operują na formatach danych. Przykładem jest konwersja liczb z dziesiętnych na binarne. Przykładem bardziej złożonego rozkazu redagowania jest rozkaz tłumaczenia (*Translate*, TR) stosowany w S/390. Może on być użyty do konwersji jednego kodu 8-bitowego na inny i operuje na trzech argumentach:

TR R1, R2, L

Argument R2 zawiera adres początkowy tablicy kodów 8-bitowych. Tłumaczone są bajty L począwszy od adresu określonego w R1, przy czym każdy bajt jest zastępowany przez zawartość wpisu w tablicy indeksowanego przez ten bajt. Na przykład w celu tłumaczenia z EBCDIC na IRA najpierw tworzymy 256-bajtową tablicę w lokacjach pamięci oznaczonych – powiedzmy –1000÷10FF w notacji szesnastkowej. Tablica zawiera znaki w kodzie IRA w ciągu według reprezentacji binarnej kodu EBCDIC; oznacza to, że kod IRA jest umieszczany w tablicy w położeniu względnym równym binarnej wartości kodu EBCDIC tego samego znaku. Wobec tego, lokacje 10F0 do 10F9 będą zawierały wartości od 30 do 39, ponieważ F0 jest kodem EBCDIC cyfry 0, a 30 jest kodem IRA tej samej cyfry (itd. aż do cyfry 9). Załóżmy teraz, że mamy kod EBCDIC dla cyfr 1984 począwszy od lokacji 2100 i chcemy go przełożyć na kod IRA. Przyjmijmy, co następuje:

- Lokacje 2100÷2103 zawierają F1 F9 F8 F4.
- R1 zawiera 2100.
- R2 zawiera 1000.

Jeśli zrealizujemy operację

TR R1, R2, 4

to lokacje 2100÷2103 będą zawierały 31 39 38 34.

Operacje wejścia-wyjścia

Rozkazy wejścia-wyjścia dość szczegółowo omówiliśmy w rozdz. 7. Jak widzieliśmy, jest stosowanych wiele rozwiązań, obejmujących izolowane programowane wejście-wyjście, programowane wejście-wyjście odwzorowane w pamięci, DMA oraz procesory wejścia-wyjścia. W wielu implementacjach przewidziano tylko niewiele rozkazów wejścia-wyjścia, przy czym poszczególne działania są określane za pomocą parametrów, kodów lub słów rozkazu.

Sterowanie systemowe

Rozkazy sterowania systemowego są na ogół rozkazami uprzywilejowanymi, które mogą być wykonywane tylko wówczas, gdy procesor znajduje się w pewnym stanie uprzywilejowanym lub gdy realizuje on program w specjalnym, uprzywilejowanym obszarze pamięci. Zwykle rozkazy te są zarezerwowane do użytku systemu operacyjnego.

Oto przykłady operacji sterowania systemowego. Rozkaz sterowania systemowego może czytać lub zmieniać zawartość rejestru sterowania; rejestry sterowania omówimy w rozdz. 12. Inny rozkaz może czytać lub modyfikować klucz ochrony pamięci, taki jak stosowany w systemie pamięciowym S/390. Jeszcze inny może przetwarzać bloki sterowania w systemie wieloprogramowym.

Przekazywanie sterowania

W przypadku wszystkich dotąd przedyskutowanych rodzajów rozkazów, domyślnie następnym rozkazem przewidzianym do wykonania jest ten, który następuje w pamięci bezpośrednio po rozkazie właśnie wykonywanym. Jednak funkcją znacznej części rozkazów w dowolnym programie jest zmiana kolejności wykonywania rozkazów. W przypadku tych rozkazów procesor aktualizuje licznik programu, aby zawierał on w pamięci adres pożądanego rozkazu.

Istnieje wiele przyczyn, dla których wymagane są operacje przekazywania sterowania. Do najważniejszych należą:

1. Podczas praktycznego używania komputerów podstawowe znaczenie ma możliwość wykonywania każdego rozkazu więcej niż raz, a być może nawet wiele tysięcy razy. Są zastosowania wymagające tysięcy lub nawet milionów rozkazów. Jest nie do pomyślenia odrębne wypisywanie każdego rozkazu. Jeśli ma być przetwarzana tablica lub lista jednostek danych, wymagana jest pętla programowa. Jedna sekwencja rozkazów jest wykonywana powtarzalnie, aż będą przetworzone wszystkie dane.
2. Praktycznie wszystkie programy obejmują podejmowanie decyzji. Chcemy, żeby komputer wykonywał określone działania w pewnych warunkach, inne zaś działania w innych warunkach. Przyjmijmy na przykład, że sekwencja rozkazów powoduje obliczanie pierwiastka kwadratowego pewnej liczby. Na początku sekwencji jest badany znak liczby. Jeśli liczba jest ujemna, obliczenia nie są przeprowadzane i sygnalizowane jest wystąpienie błędu.

3. Prawidłowe ułożenie dużego lub nawet średniego programu komputerowego jest wyjątkowo trudnym zadaniem. Jest korzystne, jeśli dysponujemy sposobami podziału takiego zadania na mniejsze części, nad którymi można pracować oddzielnie.

Omówimy teraz operacje przekazywania sterowania najczęściej spotykane w listach rozkazów: rozgałęzienie, pominięcie i wywołanie procedury.

Rozkazy rozgałęzienia

Jednym z argumentów rozkazu rozgałęzienia (zwanego również rozkazem skoku) jest adres następnego rozkazu przewidzianego do wykonania. Najczęściej mamy do czynienia z rozkazem *rozgałęzienia warunkowego*. Oznacza to, że rozgałęzienie (aktualizacja licznika programu w postaci wyrównania z adresem występującym w argumencie) jest wykonywane tylko wtedy, kiedy jest spełniony pewien warunek. W przeciwnym razie jest wykonywany następny rozkaz w sekwencji (następuje jak zwykle inkrementacja stanu licznika programu).

Istnieją dwie powszechnie stosowane metody generowania warunków, które mają być testowane w rozkazach rozgałęzienia warunkowego. Po pierwsze, większość maszyn przewiduje 1-bitowy lub wielobitowy kod warunkowy, który jest ustalany jako wynik pewnych operacji. Na przykład operacja arytmetyczna (DODAJ, ODEJMIJ itd.) może ustalić 2-bitowy kod warunkowy, który może przyjmować jedną z czterech wartości: 0, dodatni, ujemny, przepełnienie. W takiej maszynie mogłyby występować cztery różne rozkazy rozgałęzienia warunkowego:

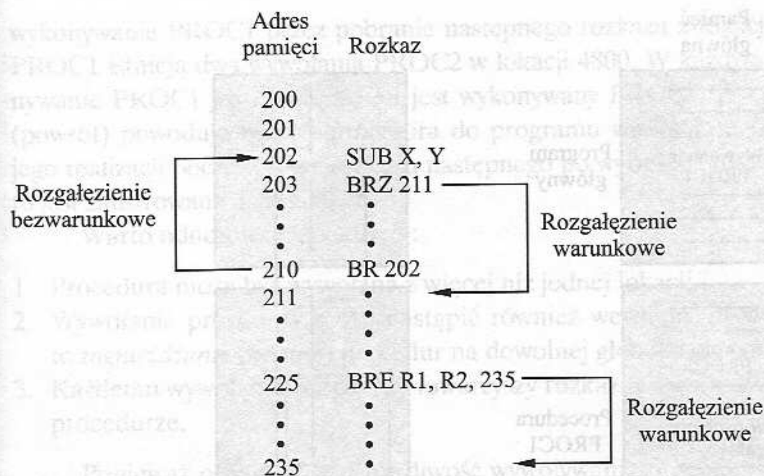
BRP X	Rozgałęzienie do lokacji X, jeśli wynik jest dodatni.
BRN X	Rozgałęzienie do lokacji X, jeśli wynik jest ujemny.
BRZ X	Rozgałęzienie do lokacji X, jeśli wynik jest zerem.
BRO X	Rozgałęzienie do lokacji X, jeśli występuje przepełnienie.

We wszystkich tych przypadkach wynik, o którym mowa, jest rezultatem ostatnio wykonanej operacji, w ramach której został też ustalony kod warunku.

Innym rozwiązaniem, które może występować łącznie z 3-adresowym formatem rozkazu, jest przeprowadzenie porównania i określenie rozgałęzienia w ramach jednego rozkazu. Na przykład:

BRE R1, R2, X Rozgałęzienie do X, jeśli zawartość R1 = zawartość R2.

Na rysunku 10.6 są pokazane przykłady tych operacji. Zauważmy, że rozgałęzienie może następować *do przodu* (do rozkazu o wyższym adresie) lub *wstecz* (do niższego adresu). Przykład pokazuje, jak można użyć rozgałęzień bezwarunkowych i warunkowych w celu utworzenia powtarzającej się pętli rozkazów. Rozkazy w lokacjach 202 do 210 będą wykonywane powtarzalnie, aż wynikiem odejmowania Y od X będzie 0.



Rysunek 10.6. Rozkazy rozgałęzienia

Rozkazy pominięcia

Inną często spotykaną formą rozkazu przekazania sterowania jest rozkaz pominięcia. Rozkaz pominięcia zawiera adres domyślny. Zwykle rozkaz pominięcia oznacza pominięcie jednego rozkazu; wobec tego adres domyślny jest równy adresowi następnego rozkazu plus długość jednego rozkazu.

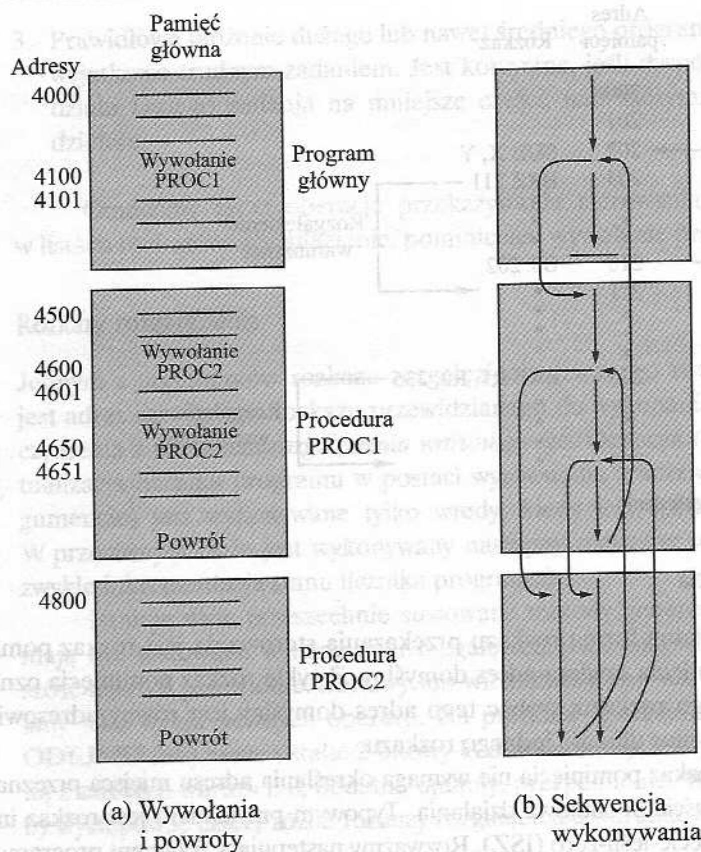
Ponieważ rozkaz pominięcia nie wymaga określania adresu miejsca przeznaczenia, może on zmieścić dodatkowe działania. Typowym przykładem jest rozkaz inkrementacja-i-pominięcie-jeśli-zero (ISZ). Rozważmy następujący fragment programu:

```
301
•
•
•
309 ISZ R1
310 BR 301
311
```

W tym fragmencie użyto dwóch rozkazów przekazania sterowania w celu zrealizowania pętli iteracyjnej. Wartość R1 jest ustalana za pomocą zanegowanej liczby iteracji, które mają być przeprowadzone. Na jednym końcu pętli następuje inkrementacja R1. Jeśli R1 nie jest równe zeru, to program rozgałęzia się do tyłu, na początek pętli. W przeciwnym razie rozgałęzienie jest pomijane, a program jest kontynuowany przez wykonywanie rozkazu następującego po zakończeniu pętli.

Rozkazy wywołania procedury

Być może najważniejszą innowacją w rozwoju języków programowania jest *procedura* (*procedure*). Procedura jest stanowiącym całość programem komputerowym, który jest wbudowywany do większego programu. Może ona być *wywołana* w dowolnym



Rysunek 10.7. Procedury zagnieżdżone

punkcie programu. Oznacza to, że w tym punkcie rozkazuje się komputerowi zrealizowanie całego podprogramu standardowego, a następnie powrót do punktu, w którym miało miejsce wywołanie.

Dwoma zasadniczymi powodami stosowania procedur są ekonomia i modułowość. Dzięki procedurze ten sam fragment kodu może być użyty wielokrotnie. Jest to ważne dla ekonomii programowania oraz dla efektywniejszego wykorzystania przestrzeni pamięci w systemie (program musi być przechowywany). Procedury umożliwiają również podział dużych zadań programowania na mniejsze części. Tak rozumiana *modułowość* znacznie ułatwia programowanie.

Mechanizm procedury obejmuje dwa podstawowe rozkazy: rozkaz wywołania, który powoduje skok z aktualnej lokacji do procedury, oraz rozkaz powrotu powodujący powrót od procedury do miejsca, w którym nastąpiło wywołanie. Oba te rozkazy należą do rozkazów rozgałęzień.

Na rysunku 10.7a jest pokazane użycie procedur do budowy programu. W tym przykładzie istnieje główny program, rozpoczynający się od lokacji 4000. Program ten zawiera wywołanie procedury PROC1, rozpoczynającej się od lokacji 4500. Gdy następuje rozkaz wywołania, procesor zawiesza realizację programu głównego i rozpoczyna

wykonywanie PROC1 przez pobranie następnego rozkazu z lokacji 4500. Wewnątrz PROC1 istnieją dwa wywołania PROC2 w lokacji 4800. W każdym przypadku wykonywanie PROC1 jest zawieszane i jest wykonywany PROC2. Dyrektywa RETURN (powrót) powoduje powrót procesora do programu wywołującego i kontynuowanie jego realizacji począwszy od rozkazu następnego po wywołaniu (CALL). Zachowanie to jest zilustrowane na rys. 10.7b.

Warto odnotować kilka uwag:

1. Procedura może być wywołana z więcej niż jednej lokacji.
2. Wywołanie procedury może nastąpić również wewnątrz procedury. Umożliwia to *zagnieżdżanie* (*nesting*) procedur na dowolnej głębokości.
3. Każdemu wywołaniu procedury towarzyszy rozkaz powrotu zawarty w wywołanej procedurze.

Ponieważ pożądana jest możliwość wywoływania procedury z wielu punktów, procesor musi zachowywać adres powrotu. Najczęściej używane są trzy miejsca przechowywania adresu powrotu:

- rejestr,
- początek wywoływanej procedury,
- wierzchołek stosu.

Rozważmy rozkaz w języku maszynowym CALL X, który oznacza *wywołaj procedury w lokacji X*. Jeśli używa się rejestru, CALL X powoduje następujące działania:

$$RN \leftarrow PC + \Delta$$

$$PC \leftarrow X$$

gdzie RN jest rejestrem, który zawsze jest używany do tego celu, PC jest licznikiem programu, a Δ – długością rozkazu. Wywołana procedura może teraz zachować zawartość RN do użycia przy późniejszym powrocie.

Drugą możliwością jest przechowywanie adresu powrotu na początku procedury. W tym przypadku CALL X powoduje:

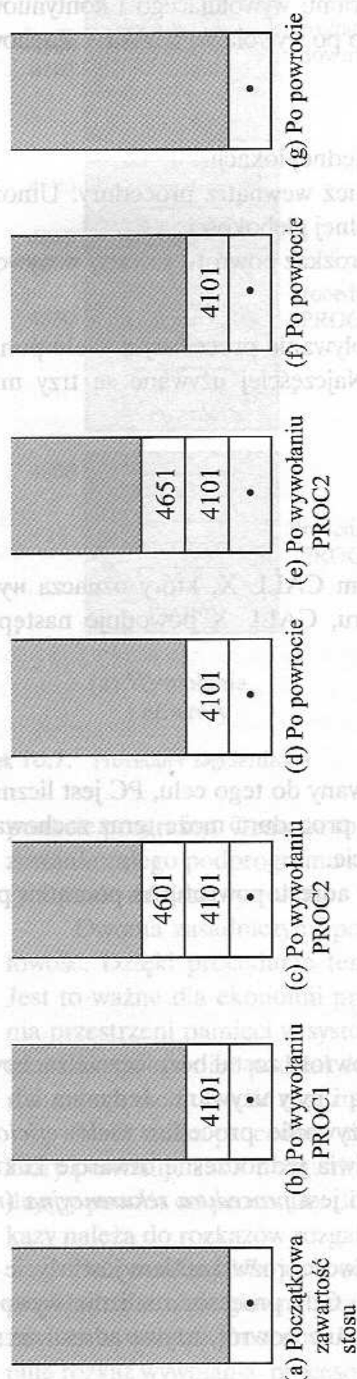
$$X \leftarrow PC + \Delta$$

$$PC \leftarrow X + 1$$

Jest to całkiem poręczne rozwiązanie. Adres powrotu został bezpiecznie zachowany.

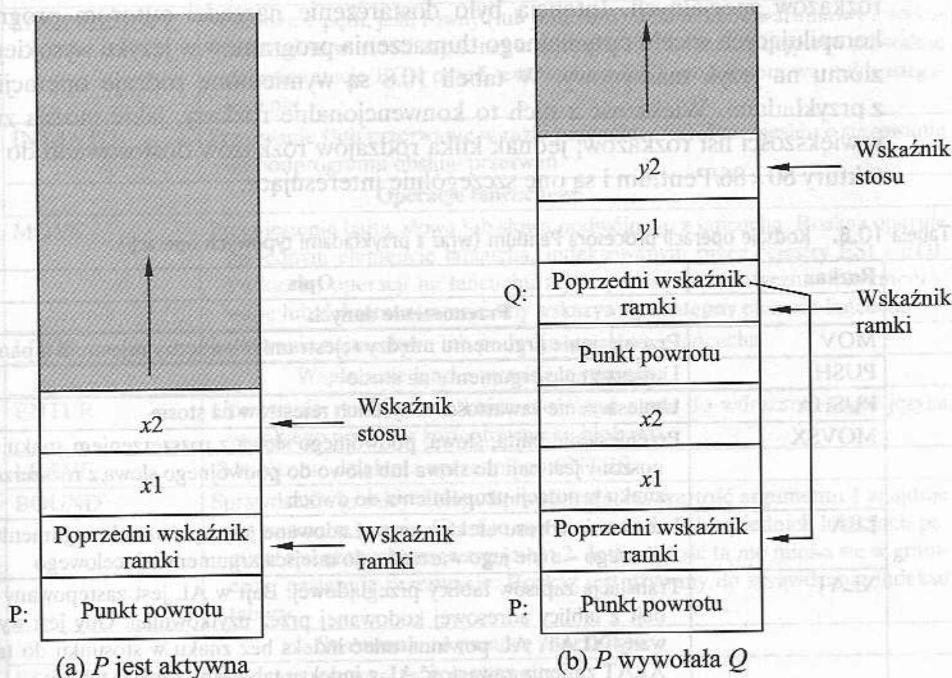
Oba przedstawione rozwiązania działają i były używane. Jedynym ich ograniczeniem jest to, że uniemożliwiają one używanie procedur *wielowejściowych* (*reentrant*). Procedura wielowejściowa umożliwia jednoczesne otwarcie kilku wywołań. Przykładem zastosowania tej własności jest *procedura rekurencyjna* (*recursive procedure*).

Ogólniejszym i stwarzającym wiele możliwości rozwiązaniem jest użycie stosu (definicja stosu znajduje się w dodatku 10A). Gdy procesor realizuje wywołanie, umieszcza adres powrotu na stosie. Gdy wykonuje powrót, używa adresu ze stosu. Na rysunku 10.8 jest pokazane użycie stosu.



Rysunek 10.8. Użycie stosu do implementacji zagnieżdżonych podprogramów z rys. 10.7

Poza zachowaniem adresu powrotu często wraz z wywołaniem procedury jest konieczne również przekazywanie parametrów. Mogą one być przekazywane w rejestrach. Inną możliwością jest przechowywanie parametrów w pamięci bezpośrednio po rozkazie wywołania. W takim przypadku powrót musi mieć miejsce do lokacji następującej po parametrach. Oba te rozwiązania mają wady. Jeśli używane są rejestry, program wywołany i program wywołujący muszą być napisane tak, żeby rejestry były właściwie używane. Przechowywanie parametrów w pamięci utrudnia wymianę zmiennej liczby parametrów. Poza tym oba rozwiązania uniemożliwiają użycie procedur wielowejściowych.



Rysunek 10.9. Zwiększenie ramki stosu przy zastosowaniu przykładowych procedur P i Q

Wykorzystując stos, można opracować bardziej elastyczne podejście do przekazywania parametrów. Gdy procesor realizuje wywołanie, przekazuje na stos nie tylko adres powrotu, ale także parametry, które mają być przekazane do pocedury. Wywołana procedura może mieć dostęp do parametrów ze stosu. W czasie powrotu parametry powrotne też mogą być umieszczone na stosie, pod adresem powrotu. Cały zestaw parametrów, łącznie z adresem powrotu, który jest przechowywany w celu wywołania procedury, określany jest jako *ramka stosu* (*stack frame*).

Przykład takiego rozwiązania jest pokazany na rys. 10.9. Odnosi się on do procedury P, w której są opisane zmienne lokalne *x*1 i *x*2, oraz procedury Q, która może być wywołana przez P i w której są opisane zmienne lokalne *y*1 i *y*2. Na tym rysunku punktem powrotu każdej procedury jest pierwsza jednostka przechowywana

w odpowiedniej ramce stosu. Następną przechowywaną jednostką jest wskaźnik początku poprzedniej ramki. Jest to konieczne, jeśli liczba parametrów na stosie lub ich długość są zmienne.

10.5. Rodzaje operacji Pentium i PowerPC

Rodzaje operacji Pentium

Dla Pentium przewidziano złożony zestaw rodzajów operacji, łącznie z pewną liczbą rozkazów specjalnych. Intencją było dostarczenie narzędzi autorom programów kompilujących w celu optymalnego tłumaczenia programów w języku wysokiego poziomu na język maszynowy. W tabeli 10.8 są wymienione rodzaje operacji wraz z przykładami. Większość z nich to konwencjonalne rozkazy, jakie można znaleźć w większości list rozkazów; jednak kilka rodzajów rozkazów dostosowano do architektury 80 × 86/Pentium i są one szczególnie interesujące.

Tabela 10.8. Rodzaje operacji procesora Pentium (wraz z przykładami typowych operacji)

Rozkaz	Opis
	Przenoszenie danych
MOV	Przeniesienie argumentu między rejestrami lub między rejestrem a pamięcią
PUSH	Umieszczenie argumentu na stosie
PUSHA	Umieszczenie zawartości wszystkich rejestrów na stosie
MOVSX	Przeniesienie bajta, słowa, podwójnego słowa z rozszerzeniem znaku. Przenoszony jest bajt do słowa lub słowo do podwójnego słowa z rozszerzeniem znaku w notacji uzupełnienia do dwóch
LEA	Ładowanie adresu efektywnego. Ładowane jest wyrównanie argumentu źródłowego – a nie jego wartość – do miejsca argumentu docelowego
XLAT	Translacja zapisów tablicy przeglądowej. Bajt w AL jest zastępowany przez bajt z tablicy adresowej kodowanej przez użytkownika. Gdy jest wykonywany XLAT, AL powinna mieć indeks bez znaku w stosunku do tablicy. XLAT zmienia zawartość AL z indeksu tablicy na zapis w tablicy
IN, OUT	Wejście lub wyjście argumentu z przestrzeni wejścia-wyjścia
	Arytmetyczne
ADD	Obliczenie sumy argumentów
SUB	Obliczenie różnicy argumentów
MUL	Obliczanie iloczynu liczb bez znaku z argumentami o długości bajta, słowa lub podwójnego słowa; wynik ma długość słowa, podwójnego słowa lub poczwórnego
IDIV	Obliczenie ilorazu
	Logiczne
AND	Wykonanie operacji AND na argumentach
BTS	Testowanie i ustawienie bitu. Operuje na polu bitowym argumentu. Kopiowana jest bieżąca wartość bitu do znacznika CF, a oryginalny bit jest ustawiany na 1
BSF	Sprawdzanie bitów w przód. Sprawdzane jest słowo lub podwójne słowo w poszukiwaniu bitu 1, a numer pierwszego bitu 1 jest zapisywany w rejestrze

Tabela 10.8 (cd.)

SHL/SHR	Przesunięcie logiczne w lewo lub w prawo
SAL/SAR	Przesunięcie arytmetyczne w lewo lub w prawo
ROL/ROR	Wykonanie rotacji w lewo lub w prawo
SETcc	Ustawienie bajta na zero lub na 1 zależnie od dowolnego z 16 warunków określonych przez znaczniki stanu
	Przekazywanie sterowania
JMP	Skok bezwarunkowy
CALL	Przeniesienie sterowania do innej lokacji. Przed przeniesieniem adres rozkazu następującego po CALL jest umieszczany na stosie
JE/JZ	Wykonanie skoku, jeśli równy (lub jeśli zero)
LOOPE/LOOPZ	Wykonanie pętli, jeśli równy (lub jeśli zero). Jest to skok warunkowy z wykorzystaniem wartości zapisanej w rejestrze ECX. Rozkaz najpierw powoduje dekrementację ECX przed testowaniem ECX pod kątem warunku rozgałęzienia
INT/INTO	Przerwanie (lub przerwanie w razie przepełnienia). Przeniesienie sterowania do podprogramu obsługi przerwań
	Operacje łańcuchowe
MOVS	Przeniesienie bajta, słowa lub słowa podwójnego z łańcucha. Rozkaz operuje na jednym elemencie łańcucha, indeksowanym przez rejestry ESI i EDI. Po każdej operacji na łańcuchu rejestry te są automatycznie inkrementowane lub dekrementowane, aby wskazywały następny element łańcucha
LODS	Ładowanie bajta, słowa lub podwójnego słowa z łańcucha
	Wspieranie języka wysokiego poziomu
ENTER	Utworzenie ramki stosu, która może być użyta do wdrażania reguł języka wysokiego poziomu zorientowanego blokowo
LEAVE	Odwrócenie działania uprzedniego ENTER
BOUND	Sprawdzenie granicy tablicy. Sprawdzenie, czy wartość argumentu 1 znajduje się wewnątrz granic. Granice są zawarte w dwóch sąsiednich lokacjach pamięci wskazanych przez argument 2. Jeśli wartość ta nie mieści się w granicach, następuje przerwanie. Rozkaz jest używany do sprawdzania indeksu tablicy
	Sterowanie za pomocą znaczników
STC	Ustawienie znacznika przeniesienia
LAHF	Ładowanie rejestru A ze znaczników stanu. Kopiuje bity SF, ZF, AF, PF i CF do rejestru A
	Rejestr segmentowy
LDS	Ładowanie wskaźnika do rejestru segmentowego D
HLT	Zatrzymanie
LOCK	Potwierdzenie utrzymania pamięci wspólnej. Dzięki temu Pentium ma wyłączne prawo używania jej podczas rozkazu, który następuje bezpośrednio po LOCK
ESC	Ucieczka związana z rozszerzeniem procesora. Kod ucieczki, który wskazuje, że następne rozkazy mają być wykonywane przez koprocesor numeryczny, wspierający obliczenia całkowitoliczbowe i zmiennopozycyjne o dużej dokładności
WAIT	Czekanie, aż BUSY# będzie zanegowane. Wykonywanie programu jest zawieszane przez Pentium, aż procesor wykryje, że końcówka BUSY nie jest aktywna, co wskazuje, że koprocesor numeryczny zakończył obliczenia

Tabela 10.8 (cd.)

Ochrona	
SGDT	Zapisanie globalnej tablicy deskryptorów
LSL	Ładowanie granicy segmentu do rejestru określonego przez użytkownika
VERR/VERW	Weryfikowanie segmentu w przypadku odczytu/zapisu
Zarządzanie pamięcią podręczną	
INVD	Opróżnienie wewnętrznej pamięci podręcznej
WBINVD	Opróżnienie wewnętrznej pamięci podręcznej po zapisaniu w pamięci „zanieczyszczonych" wierszy
INVLPG	Unieważnienie zapisu w buforze translacji adresów tablic stron

Rozkazy wywołania/powrotu

Pentium przewiduje cztery rozkazy obsługujące procedurę wywołania i powrotu: CALL, ENTER, LEAVE i RETURN. Pouczające jest zapoznanie się z ich funkcjonowaniem. Przypomnijmy sobie na podstawie rys. 10.9, że typowym środkiem wdrażania procedury wywołanie-powrót jest użycie ramek stosu. Gdy wywoływana jest nowa procedura, podczas przechodzenia do nowej procedury muszą być wykonane następujące działania:

- Umieszczenie punktu powrotu na stosie.
- Umieszczenie bieżącego wskaźnika ramki na stosie.
- Skopiowanie wskaźnika stosu jako nowej wartości wskaźnika ramki.
- Skorygowanie wskaźnika stosu w celu alokacji ramki.

Rozkaz CALL powoduje umieszczenie bieżącej wartości wskaźnika rozkazu na stosie oraz skok do początkowego punktu procedury przez umieszczenie adresu tego punktu we wskaźniku rozkazów. W maszynach 8088 i 8086 typowa procedura rozpoczynała się od sekwencji

```
PUSH  EBP
MOV   EBP, ESP
SUB   ESP, space_ for_locals
```

gdzie EBP jest wskaźnikiem ramki, a ESP – wskaźnikiem stosu. W procesorze 80286 i w późniejszych procesorach rozkaz ENTER powodował wykonanie wszystkich powyższych operacji za pomocą jednego rozkazu.

Rozkaz ENTER został wprowadzony do listy rozkazów w celu zapewnienia bezpośredniego wsparcia kompilatora. Umożliwia on realizację tzw. *procedur zagnieżdżonych* (*nested procedures*) w takich językach, jak Pascal, Cobol i Ada (nie występujących w C i w Fortranie). Okazało się, że istnieją lepsze sposoby postępowania z procedurami zagnieżdżonymi dla tych właśnie języków. Ponadto, chociaż rozkaz ENTER oszczędza kilka bajtów pamięci w porównaniu z sekwencją PUSH, MOV i SUB (4 bajty wobec 6 bajtów), w rzeczywistości wydłuża czas realizacji (10 cykli zegara wobec 6 cyklu). Chociaż więc wydawało się dobrym pomysłem uzu-

pełnienie listy rozkazów przez ENTER, skomplikowało to implementację procesora, wnosząc przy tym niewielkie lub żadne korzyści. Zobaczymy, że w przeciwieństwie do tego rozwiązanie RISC przy projektowaniu procesora zapobiega tak złożonym rozkazom, jak ENTER, umożliwiając przy tym efektywniejszą implementację za pomocą sekwencji prostszych rozkazów.

Zarządzanie pamięcią

Inny zestaw rozkazów specjalnych dotyczy segmentowania pamięci. Są to rozkazy uprzywilejowane, które mogą być wykonywane tylko przez system operacyjny. Umożliwiają one ładowanie i czytanie lokalnych oraz globalnych tablic segmentów (zwanych tablicami deskryptorów) oraz na sprawdzanie i zmienianie poziomu uprzywilejowania segmentów.

Rozkazy specjalne dotyczące wewnętrznej pamięci podręcznej zostały omówione w rozdz. 4.

Kody warunkowe

Wiemy już, że kody warunkowe są bitami w specjalnych rejestrach, które mogą być ustawiane przez pewne operacje i używane w rozkazach rozgałęzienia warunkowego. Warunki te są ustalane przez operacje arytmetyczne i porównania. Operacja porównania w większości języków polega na odjęciu dwóch argumentów, podobnie jak w operacji odejmowania. Różnica polega na tym, że operacja porównania ustala tylko kody warunkowe, podczas gdy operacja odejmowania zapisuje również wynik odejmowania w miejscu przeznaczenia.

Tabela 10.9. Kody warunkowe procesora Pentium

Bit stanu	Nazwa	Opis
C	Carry	Wskazuje przeniesienie lub przeniesienie zanegowane do lewej pozycji bitowej w wyniku operacji arytmetycznej. Modyfikowany również przez niektóre operacje przesunięcia lub rotacji
P	Parity	Parzystość wyniku operacji arytmetycznej lub logicznej. 1 wskazuje parzystość, 0 – nieparzystość
A	Auxiliary Carry	Reprezentuje przeniesienie lub przeniesienie zanegowane między półbajtami w wyniku 8-bitowej operacji arytmetycznej lub logicznej z użyciem rejestru AL
Z	Zero	Wskazuje, że wynik operacji arytmetycznej lub logicznej jest równy 0
S	Sign	Wskazuje znak wyniku operacji arytmetycznej lub logicznej
O	Overflow	Wskazuje przepełnienie arytmetyczne po dodawaniu lub odejmowaniu

W tabeli 10.9 są wymienione kody warunkowe stosowane w Pentium. Każdy warunek lub kombinacja warunków mogą być testowane pod kątem rozgałęzienia warunkowego. W tabeli 10.10 są podane kombinacje warunków, dla których zdefiniowano kody operacyjne rozgałęzień warunkowych.

Tabela 10.10. Stosowane w Pentium warunki dla skoków warunkowych i rozkazów SETcc

Symbol	Testowany warunek	Komentarz
A, NBE	C=0 AND Z=0	Ponad; nie poniżej lub równe (większa niż, bez znaku)
AE, NB, NC	C=0	Ponad lub równe; nie poniżej (większa niż lub równa, bez znaku); nie przeniesienie
B, NAE, C	C=1	Poniżej; nie powyżej lub równe (mniejsza niż, bez znaku); ustawione przeniesienie
BE, NA	C=1 OR Z=1	Poniżej lub równe; nie powyżej (mniejsza lub równa, bez znaku)
E, Z	Z=1	Równe; zero (ze znakiem lub bez znaku)
G, NLE	[(S=1 AND O=1) OR (S=0 AND O=0)] AND [Z=0]	Większa niż; nie mniejsza lub równa (ze znakiem)
GE, NL	(S=1 AND O=1) OR (S=0 AND O=0)	Większa niż lub równa; nie mniejsza niż (ze znakiem)
L, NGE	(S=1 AND O=0) OR (S=0 AND O=1)	Mniejsza niż; nie większa lub równa (ze znakiem)
LE, NG	(S=1 AND O=0) OR (S=0 AND O=1) OR (Z=1)	Mniejsza niż lub równa; nie większa (ze znakiem)
NE, NZ	Z=0	Nie równa; nie zero (ze znakiem lub bez znaku)
NO	O=0	Brak przepełnienia
NS	S=0	Nie znak (nie ujemna)
NP, PO	P=0	Brak parzystości; nieparzystość
O	O=1	Przepełnienie
P	P=1	Parzystość
S	S=1	Znak (ujemna)

Na podstawie tej listy można poczynić kilka interesujących spostrzeżeń. Po pierwsze, może być potrzebne sprawdzenie dwóch argumentów w celu stwierdzenia, czy jedna liczba jest większa od drugiej. Zależy to jednak od tego, czy liczby te mają znaki. Na przykład, 8-bitowa liczba 11111111 jest większa od 00000000, jeśli obie liczby są interpretowane jako bezznakowe liczby całkowite (255 > 0), jest jednak mniejsza, jeśli są traktowane jako 8-bitowe liczby w notacji uzupełnienia do dwóch (–1 < 0). Wiele języków asemblerowych wprowadza dlatego dwa zestawy wyrażeń w celu odróżnienia obu przypadków: jeśli porównujemy dwie liczby jako liczby całkowite ze znakiem, używamy wyrażeń *większa niż* lub *mniejsza niż*; jeśli natomiast porównujemy je jako liczby całkowite bezznakowe, używamy wyrażeń *poniżej* i *ponad*.

Druga obserwacja wiąże się ze złożonością porównywania liczb całkowitych ze znakami. Wynik ze znakiem jest większy lub równy zeru, jeżeli (a) bit znaku jest zerem i nie ma przepełnienia (S=0 oraz O=0), lub (b) bit znaku jest jedynką i występuje przepełnienie. Analiza rysunku 9.4 powinna przekonać czytelnika, że warunki przetestowane dla różnych operacji ze znakami są właściwe (zobacz problem 10.14).

Rozkazy Pentium MMX

W roku 1996 firma Intel wprowadziła do swojego szeregu produktów Pentium technologię MMX. Jest ona zbiorem wysoce zoptymalizowanych rozkazów przeznaczonych do zadań multimedialnych. Istnieje 57 nowych rozkazów, które traktują dane w sposób określany jako SIMD (*single-instruction, multiple-data*, jeden rozkaz, wiele danych). Umożliwia to wykonywanie takich samych operacji – na przykład mnożenia lub dodawania – jednocześnie na wielu danych. Wykonanie każdego rozkazu zajmuje zwykle jeden cykl. W określonych aplikacjach tego rodzaju szybkie, równoległe operacje mogą spowodować 2÷8-krotne przyspieszenie w zestawieniu z porównywalnymi algorytmami nie korzystającymi z rozkazów MMX [ATKI96].

Technologia MMX koncentruje się na programowaniu multimediów. Dane wideo i audio składają się zwykle z wielkich tablic danych o małych rozmiarach, takich jak 8- lub 16-bitowe, podczas gdy rozkazy konwencjonalne są dostosowane do operowania na danych 32- lub 64-bitowych. Oto wybrane przykłady: w grafice i w wideo pojedyncza scena składa się z tablicy pikseli[1], a każdemu pikselowi lub każdemu składnikowi koloru (czerwonemu, niebieskiemu, zielonemu) piksela odpowiada 8 bitów. Typowym próbkom audio przypisuje się 16 bitów. W przypadku pewnych algorytmów związanych z grafiką trójwymiarową powszechnie występują dane o długości 32 bitów. W celu umożliwienia równoległego realizowania operacji na danych o tej długości w MMX zostały zdefiniowane trzy nowe rodzaje danych. Każdy z tych rodzajów danych ma długość 64 bitów i składa się z wielu mniejszych pól danych, z których każde obejmuje stałopozycyjną liczbę całkowitą. Rodzaje te są następujące:

- **Upakowany bajt.** Osiem bajtów upakowanych w postaci wielkości 64-bitowej.
- **Upakowane słowo.** Cztery 16-bitowe słowa upakowane w postaci 64 bitów.
- **Upakowane podwójne słowo.** Dwa 32-bitowe słowa podwójne upakowane w postaci 64 bitów.

W tabeli 10.11 została przedstawiona lista rozkazów MMX. Większość rozkazów obejmuje równoległe operacje na bajtach, słowach lub słowach podwójnych. Na przykład rozkaz PSLLW realizuje przesunięcie logiczne w lewo oddzielnie na każdym spośród czterech słów w upakowanym słowie argumentu; rozkaz PADDB przyjmuje jako wejście upakowane argumenty bajtowe i wykonuje równoległe, niezależne dodawania każdej pozycji bajtowej, dając wynik w postaci bajtowo upakowanych danych wyjściowych.

Jedną z niezwykłych właściwości tej nowej listy rozkazów jest wprowadzenie arytmetyki nasycenia. Gdy w zwykłej arytmetyce bezznakowej określona operacja prowadzi do przepełnienia (tzn. przeniesienia najbardziej znaczącego bitu), dodat-

[1] Piksel (nazwa pochodzi od *picture element*) jest najmniejszym elementem obrazu cyfrowego, któremu można przypisać poziom szarości. Określenie równoważne: piksel jest pojedynczym punktem w punktowo-matrycowej reprezentacji obrazu.

Tabela 10.11. Lista rozkazów MMX

Kategoria	Rozkaz	Opis
Arytmetyczne	PADD[B, W, D]	Równoległe dodawanie 8 upakowanych bajtów, cztery 16-bitowe słowa lub dwa 32-bitowe słowa podwójne, z zawinięciem
	PADDS [B, W]	Dodawanie z nasyceniem
	PADDUS [B, W]	Dodawanie bezznakowe z nasyceniem
	PSUB [B, W, D]	Odejmowanie z zawinięciem
	PSUBS [B, W]	Odejmowanie z nasyceniem
	PSUBUS [B, W]	Odejmowanie bezznakowe z nasyceniem
	PMULHW	Równoległe mnożenie czterech 16-bitowych słów ze znakami z wybranymi 16 bitami wyższego rzędu z 32-bitowego wyniku
	PMULLW	Równoległe mnożenie czterech 16-bitowych słów ze znakami z wybranymi 16 bitami niższego rzędu z 32-bitowego wyniku
	PMADDWD	Równoległe mnożenie czterech 16-bitowych słów ze znakami; dodawanie sąsiadujących par z 32-bitowych wyników
Porównanie	PCMPEQ [B, W, D]	Porównywanie równoległe pod kątem równości; wynikiem jest maskowanie jedynek, jeśli prawda, lub zer, jeśli fałsz
	PCMPGT [B, W, D]	Porównywanie równoległe pod kątem „większy niż"; wynikiem jest maskowanie jedynek, jeśli prawda, lub zer, jeśli fałsz
Konwersja	PACKUSWB	Upakowanie słów w bajty z bezznakowym nasyceniem
	PACKSS [WB, DW]	Upakowanie słów w bajty lub podwójnych słów w słowa z nasyceniem i ze znakiem
	PUNPCKH [BW, WD, DQ]	Równoległe rozpakowywanie (łączenie przeplatane) bajtów wysokiego rzędu, słów lub podwójnych słów z rejestru MMX
	PUNPCKL [BW, WD, DQ]	Równoległe rozpakowywanie (łączenie przeplatane) bajtów niskiego rzędu, słów lub podwójnych słów z rejestru MMX
Logiczne	PAND	64-bitowe logiczne AND na poziomie bitowym
	PNDN	64-bitowe logiczne AND NOT na poziomie bitowym
	POR	64-bitowe logiczne OR na poziomie bitowym
	PXOR	64-bitowe logiczne XOR na poziomie bitowym
Przesunięcie	PLL [W, D, Q]	Równoległe logiczne przesunięcie w lewo upakowanych słów, podwójnych słów lub poczwórnych słów o wielkość określoną w rejestrze MMX lub o wartość natychmiastową
	PSRL [W, D, Q]	Równoległe logiczne przesunięcie w prawo upakowanych słów, podwójnych słów lub poczwórnych słów
	PSRA [W, D]	Równoległe arytmetyczne przesunięcie w prawo upakowanych słów, podwójnych słów lub poczwórnych słów
Transfer danych	MOV [D, Q]	Przeniesienie słowa podwójnego lub poczwórnego do (lub z) rejestru MMX
Zarządzanie stanem	EMMS	Opróżnienie stanu MMX (opróżnienie bitów znacznika rejestrów zmiennopozycyjnych)

Uwaga: Jeśli dany rozkaz obsługuje wiele rodzajów danych [bajty (B), słowa (W), podwójne słowa (D), poczwórne słowa (Q)], rodzaje danych są wskazane w nawiasach.

kowy bit jest obcinany. Jest to określane jako zawinięcie, ponieważ efektem obcięcia może być na przykład wynik dodawania mniejszy od dwóch argumentów wejściowych. Rozważmy dodawanie dwóch słów w notacji szesnastkowej, F000h i 3000h. Ich suma może być wyrażona jako

```
 F000h = 1111 0000 0000 0000
+3000h = 0011 0000 0000 0000
        10010 0000 0000 0000 = 2000h
```

Gdyby te dwie liczby reprezentowały intensywność obrazu, to wynikiem dodawania byłby taki, że kombinacja dwóch silnych cieni okazałaby się jaśniejsza. Zwykle taki wynik jest niepożądany. Jeśli w arytmetyce nasycenia dodawanie prowadzi do przepełnienia lub odejmowanie do niedomiaru, wynik jest ustawiany jako największa lub najmniejsza wartość reprezentowalna. W takiej arytmetyce poprzedni przykład wygląda następująco:

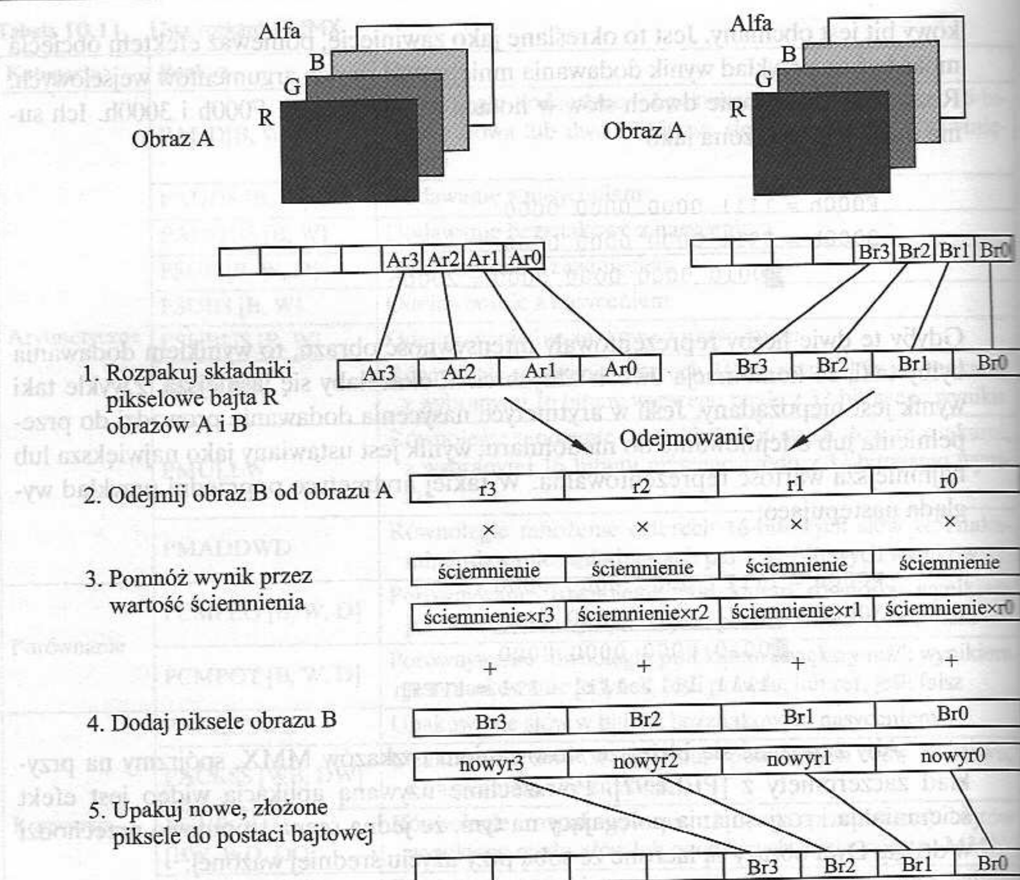
```
 F000h = 1111  0000 0000 0000
+3000h = 0011  0000 0000 0000
        10010 0000 0000 0000
         1111 1111 1111 1111 = FFFFh
```

Aby zapoznać się bliżej ze stosowaniem rozkazów MMX, spójrzmy na przykład zaczerpnięty z [PELE97]. Powszechnie używaną aplikacją wideo jest efekt ściemniania i rozjaśniania polegający na tym, że jedna scena stopniowo przechodzi w drugą. Dwa obrazy są łączone ze sobą przy użyciu średniej ważonej:

$$\text{Piksel_wynikowy} = \text{Piksel_A} \times \text{ściemnianie} + \text{Piksel_B} \times (1 - \text{ściemnianie})$$

Obliczenie to jest dokonywane na każdej pozycji pikselowej w A i B. Jeśli tworzony jest szereg ramek wideo o stopniowo zmieniającej się wartości ściemnienia od 1 do 0 (skalowanej odpowiednio w odniesieniu do 8-bitowej liczby całkowitej), wynikiem jest stopniowe przejście od obrazu A do obrazu B.

Na rysunku 10.10 jest pokazana sekwencja kroków wymagana w odniesieniu do jednego zbioru pikseli. Ośmiobitowe składniki pikselowe są poddawane konwersji na elementy 16-bitowe w celu dostosowania się do możliwości 16-bitowego mnożenia MMX. Jeśli rozdzielczość obrazów wynosi 640×480, a w metodzie przenikania wykorzystuje się wszystkie możliwe wartości ściemnienia w liczbie 255, to łączna liczba rozkazów wykonywanych przy użyciu MMX wynosi 535 milionów. To samo obliczenie wykonywane bez rozkazów MMX wymaga 1,4 miliarda rozkazów [INTE98].



Sekwencja kodów MMX realizująca tę operację:

```
pxor       mm7, mm7     ;wyprowadzenie zer z mm7
movq       mm3, fad_val ;ładowanie wartości ściemnienia powielonej 4-krotnie
movd       mm0, imageA  ;ładowanie 4 czerwonych składników pikselowych z obrazu A
movd       mm1, imageB  ;ładowanie 4 czerwonych składników pikselowych z obrazu B
punpckblw  mm0, mm7     ;rozpakowanie 4 pikseli do 16 bitów
punpckblw  mm1, mm7     ;rozpakowanie 4 pikseli do 16 bitów
psubw      mm0, mm1     ;odejmowanie obrazu B od obrazu A
pmulhw     mm0, mm3     ;mnożenie wyników odejmowania przez wartości ściemnienia
padddw     mm0, mm1     ;dodawanie wyniku do obrazu B
packuswb   mm0, mm7     ;pakowanie 16-bitowych wyników do postaci bajtowej
```

Rysunek 10.10. Składanie płaskiego obrazu kolorowego [PELE97]

Rodzaje operacji PowerPC

PowerPC zapewnia duży zbiór rodzajów operacji. W tabeli 10.12 są wymienione rodzaje i odpowiednie przykłady. Warto zwrócić uwagę na kilka właściwości.

Tabela 10.12. Rodzaje operacji procesora PowerPC (z przykładami typowych operacji)

Rozkaz	Opis
	Zorientowane na rozgałęzienia
b	Rozgałęzienie bezwarunkowe
bl	Rozgałęzienie do adresu docelowego i umieszczenie adresu efektywnego rozkazu następującego po rozgałęzieniu w rejestrze powiązań
bc	Rozgałęzienie warunkowe na podstawie rejestru zliczającego i/lub bitu w rejestrze warunkowym
sc	Systemowe wywołanie obsługi systemu operacyjnego
trap	Porównanie dwóch argumentów i uaktywnienie programu obsługi pułapek, jeśli spełnione są określone warunki
	Ładuj/zapisz
lwzu	Załadowanie słowa i rozszerzenie zerami do lewej; zaktualizowanie rejestru źródłowego
ld	Załadowanie podwójnego słowa
lmw	Załadowanie wielokrotnego słowa; ładowanie kolejnych słów do sąsiednich rejestrów z rejestru docelowego poprzez rejestr roboczy 31
lswx	Załadowanie łańcucha bajtów do rejestrów, rozpoczynając od rejestru docelowego; cztery bajty na rejestr; zawinięcie od rejestru 31 do rejestru 0
	Arytmetyka całkowitoliczbowa
add	Obliczenie sumy zawartości dwóch rejestrów i umieszczenie w trzecim rejestrze
subf	Obliczenie różnicy zawartości dwóch rejestrów i umieszczenie w trzecim rejestrze
mullw	Obliczenie iloczynu 32-bitowych zawartości niskiego rzędu dwóch rejestrów i umieszczenie 64-bitowego iloczynu w trzecim rejestrze
divd	Obliczenie ilorazu 64-bitowej zawartości dwóch rejestrów i umieszczenie wyniku w trzecim rejestrze
	Logiczne i przesunięcia
cmp	Porównanie dwóch argumentów i ustawienie czterech bitów warunkowych w określonym polu rejestru warunkowego
crand	Wykonanie operacji AND w rejestrze warunkowym; operacja AND dotyczy dwóch bitów rejestru warunkowego, wynik jest umieszczany w jednej z dwóch pozycji bitowych
and	Wykonanie operacji AND na zawartości dwóch rejestrów i umieszczenie wyniku w trzecim rejestrze
cntlzd	Policzenie liczby kolejnych zer, poczynając od bitu zero w rejestrze źródłowym, i umieszczenie wyniku w rejestrze docelowym
rldic	Wykonanie rotacji w lewo rejestru podwójnego słowa, wykonanie operacji AND z maską i zapisanie wyniku w rejestrze docelowym
sld	Przesunięcie w lewo bitów w rejestrze źródłowym i zapisanie w rejestrze docelowym
	Zmiennopozycyjne
lfs	Załadowanie 32-bitowej liczby zmiennopozycyjnej z pamięci, przetworzenie na format 64-bitowy i zapisanie w rejestrze zmiennopozycyjnym
fadd	Obliczenie sumy zawartości dwóch rejestrów i umieszczenie wyniku w trzecim rejestrze
fmadd	Obliczenie iloczynu zawartości dwóch rejestrów, dodanie zawartości trzeciego i umieszczenie wyniku w czwartym rejestrze
fcmpu	Porównanie dwóch argumentów zmiennopozycyjnych i ustawienie bitów warunkowych
	Zarządzanie pamięcią podręczną
dcbf	Opróżnienie bloku pamięci podręcznej danych; wykonanie przeglądu pamięci podręcznej pod kątem określonego adresu docelowego i przeprowadzenie operacji opróżniania
icbi	Unieważnienie bloku pamięci podręcznej rozkazów

Rozkazy dotyczące rozgałęzień

PowerPC obsługuje zwykłe przypadki rozgałęzień bezwarunkowych i warunkowych. Warunkowe rozkazy rozgałęzienia testują pojedynczy bit rejestru warunkowego (możliwości: „prawdziwy", „fałszywy" lub „nie dbaj o to") oraz zawartość rejestru licznika (możliwości: „zero", „niezero" lub „nie dbaj o to"). Jest zatem 9 oddzielnych warunków, które mogą być definiowane dla warunkowych rozkazów rozgałęzienia. Jeśli rejestr licznika jest testowany pod kątem zero – niezero, to przed testem jego stan jest zmniejszany o 1. Jest to wygodne przy tworzeniu pętli iteracyjnych.

Rozkazy rozgałęzienia mogą także wskazywać, że adres lokacji następującej po rozgałęzieniu ma być umieszczony w rejestrze powiązania, opisanym w rozdz. 14. Ułatwia to przetwarzanie wywoływań i powrotów.

Rozkazy ładowania/zapisu

W architekturze procesora PowerPC tylko rozkazy ładowania i zapisu mają dostęp do pamięci; rozkazy arytmetyczne i logiczne są przeprowadzane tylko na rejestrach. Jest to charakterystyczne dla projektów RISC, co omówimy w rozdz. 13.

Istnieją dwie właściwości, które charakteryzują różne rozkazy ładowania i zapisu. Są to:

- **Rozmiar danych.** Jednostkami transferu danych są bajt, półsłowo, słowo i podwójne słowo. Są również osiągalne rozkazy ładowania i zapisu łańcuchów bajtów do (i z) wielu rejestrów.
- **Rozszerzenie znaku.** W przypadku ładowania półsłów i słów nie używane bity znajdujące się w lewej części 64-bitowego rejestru przeznaczenia są albo wypełniane zerami, albo bitem znaku ładowanej wielkości.

10.6. Język asemblerowy

Procesor jest w stanie rozumieć i wykonywać rozkazy maszynowe. Rozkazy takie są po prostu liczbami binarnymi przechowywanymi w komputerze. Jeśli programista chciałby programować bezpośrednio w języku maszynowym, byłoby konieczne wprowadzanie programu w postaci danych binarnych.

Rozważmy prostą instrukcję w języku BASIC:

N=I + J + K

Załóżmy, że chcemy zaprogramować tę instrukcję w języku maszynowym oraz nadać zmiennym I, J i K wartość początkową odpowiednio 2, 3 i 4, co widać na rys. 10.11a. Program rozpoczyna się od lokacji 101 (szesnastkowej). Pamięć jest zarezerwowana dla czterech zmiennych począwszy od lokacji 201. Program składa się z czterech rozkazów:

1. Ładuj zawartość lokacji 201 do AC.
2. Dodaj zawartość lokacji 202 do AC.
3. Dodaj zawartość lokacji 203 do AC.
4. Zapisz zawartość AC w lokacji 204.

Jest to oczywiście proces żmudny i podatny na błędy.

Niewielkim ulepszeniem jest napisanie programu w notacji szesnastkowej zamiast w binarnej (rys. 10.11c). Moglibyśmy napisać program jako ciąg wierszy. Każdy wiersz zawiera adres lokacji pamięci oraz kod szesnastkowy wartości binarnej, która ma być przechowana w tej lokacji. Potrzebujemy następnie programu, który zaakceptuje takie dane wejściowe, przetłumaczy każdy wiersz na liczbę binarną i zapisze ją w określonej lokacji.

Adres	Zawartość			
101	0010	0010	0000	0001
102	0001	0010	0000	0010
103	0001	0010	0000	0011
104	0011	0010	0000	0100
201	0000	0000	0000	0010
202	0000	0000	0000	0011
203	0000	0000	0000	0100
204	0000	0000	0000	0000

(a) program binarny

Adres	Zawartość	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(b) program symboliczny

Adres	Zawartość
101	201
102	202
103	203
104	204
201	0002
202	0003
203	0004
204	0000

(c) program szesnastkowy

Etykieta	Operacja	Argument
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) program asemblerowy

Rysunek 10.11. Obliczanie wyrażenia N = I + J + K

Aby uzyskać znaczną poprawę, możemy używać nazw symbolicznych (mnemoników) każdego rozkazu. Wynikiem jest *program symboliczny* pokazany na rys. 10.11b. Każdy wiersz nadal reprezentuje jedną lokację w pamięci. Każdy wiersz składa się z trzech pól oddzielonych spacjami. Pierwsze pole zawiera adres lokacji. W przypadku rozkazu drugie pole zawiera 3-literowy symbol kodu operacji. Jeśli jest to rozkaz odnoszący się do pamięci, to trzecie pole zawiera adres. W celu zapi-

sania dowolnych danych w lokacji wynaleźliśmy *pseudorozkaz* o symbolu DAT. Jest to jedynie wskazanie, że trzecie pole w wierszu zawiera liczbę szesnastkową, która ma być zapisana w lokacji określonej przez pierwsze pole.

Dla tego rodzaju danych wejściowych potrzebujemy nieco bardziej złożonego programu. Program akceptuje każdy wiersz danych wejściowych, generuje liczbę binarną opartą na drugim i trzecim (jeśli występuje) polu i zapisuje ją w lokacji określonej przez pierwsze pole.

Użycie programu symbolicznego ułatwia życie, jednak nadal jest kłopotliwe. Po pierwsze, musimy określać bezwzględne adresy każdego słowa. Oznacza to, że program i dane mogą być ładowane tylko do jednego miejsca w pamięci i że musimy znać to miejsce z wyprzedzeniem. Gorzej, jeśli pewnego dnia chcemy zmienić program, dodając lub usuwając jakiś wiersz. Zmieni to adresy wszystkich następnych słów.

O wiele lepszym, powszechnie obecnie używanym sposobem jest użycie adresów symbolicznych. Zilustrowano to na rys. 10.11d. Każdy wiersz nadal składa się z trzech pól. Pierwsze pole nadal jest przeznaczone na adres, jednak zamiast bezwzględnego adresu numerycznego jest używany symbol. Niektóre wiersze nie mają adresu, co powoduje, że adres tego wiersza jest o 1 większy niż adres poprzedniego wiersza. W przypadku adresów odnoszących się do pamięci trzecie pole również zawiera adres symboliczny.

Dzięki temu ostatniemu ulepszeniu uzyskaliśmy *język asemblerowy*. Programy napisane w języku asemblerowym (programy asemblerowe) są tłumaczone na język maszyny przez asembler. Program ten musi nie tylko tłumaczyć symbole przedyskutowane wcześniej, ale także przypisywać pewną postać adresów pamięci adresom symbolicznym.

Opracowanie języka asemblerowego było kamieniem milowym w ewolucji techniki komputerowej. Był to pierwszy krok do języków wysokiego poziomu używanych dzisiaj. Chociaż niewielu programistów używa języka asemblerowego, praktycznie wszystkie maszyny umożliwiają to. Możliwość taka jest potrzebna (jeżeli w ogóle) w przypadku programów systemowych, takich jak kompilatory i procedury wejścia-wyjścia.

10.7. Polecana literatura

Wiele książek zawiera wystarczające omówienie języka maszynowego i projektowania list rozkazów, w tym [PATT98], [TANE99] i [HAYE98]. Lista rozkazów Pentium jest omówiona w [BREY00]. Listę rozkazów PowerPC przedstawiono w [IBM94] i [WEIS94].

BREY00 Brey B.: *The Intel Microprocessors: 8086/8066, 80186/80188, 80288, 80386, 80486, Pentium, Pentium Pro i Pentium II Processors*. Upper Saddle River, Prentice Hall, 2000.

HAYE98 Hayes J.: *Computer Architecture and Organization. Wyd. II*. New York, McGraw-Hill, 1998.

IBM94 International Business Machines, Inc.: *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, Morgan-Kaufmann, 1994.

PATT98 Patterson D., Hennessy J.: *Computer Organisation and Desidn: The Hardware/Software Interface*. San Mateo, CA: Morgan Kaufmann, 1998.

TANE99 Tannenbaum A.: *Structured Computer Organization*. Englewood Cliffs, Prentice Hall, 1999.

WEIS94 Weiss S., Smith J.: *POWER and PowerPC*. San Francisco, Morgan-Kaufmann, 1994.

10.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Adres – *address*
Akumulator – *accumulator*
Argument – *operand*
Cienkokońcowy – *little endian*
Dwukońcowy – *bi-endian*
Grubokońcowy – *big endian*
Lista rozkazów – *instruction set*
Odwrotna notacja polska – *reverse Polish notation*
Operacja – *operation*
Pominięcie – *skip*
POP – zdejmować ze stosu
Powrót procedury – *procedure return*
Procedura wielowejściowa – *reentrant procedure*
Przesunięcie arytmetyczne – *arithmetic shift*
Przesunięcie logiczne – *logical shift*
PUSH – umieszczać na stosie
Rotacja – *rotate*
Rozgałęzienie – *branch*
Rozgałęzienie warunkowe – *conditional branch*
Rozkaz maszynowy – *machine instruction*
Skok – *jump*
Stos – *stack*
Upakowana notacja dziesiętna – *packed decimal*
Wywołanie procedury – *procedure call*

Pytania kontrolne

10.1. Jakie są typowe składniki rozkazu maszynowego?
10.2. Jakie rodzaje lokacji mogą zawierać argumenty źródłowe i docelowe?
10.3. Jeśli rozkaz zawiera cztery adresy, jakie może być ich przeznaczenie?
10.4. Wymień i krótko objaśnij pięć ważnych zagadnień projektowania list rozkazów.
10.5. Jakie rodzaje argumentów występują typowo na listach rozkazów maszynowych?
10.6. Jaka jest zależność między kodem znaków IRA a upakowaną reprezentacją dziesiętną?
10.7. Jaka jest różnica między przesunięciem arytmetycznym a logicznym?
10.8. Dlaczego są potrzebne rozkazy przekazywania sterowania?
10.9. Wymień i krótko objaśnij dwa powszechnie stosowane sposoby generowania warunku, który ma być sprawdzony w rozkazie rozgałęzienia warunkowego.
10.10. Jakie jest znaczenie wyrażenia *zagnieżdżanie procedur*?
10.11. Wymień trzy możliwe miejsca przechowywania adresu powrotnego w odniesieniu do powrotu procedury.
10.12. Co to jest procedura wielowejściowa?
10.13. Jaka jest różnica między językiem asemblerowym a maszynowym?
10.14. Co to jest odwrotna notacja polska?
10.15. Jaka jest różnica między stylem cienkokońcowym a grubokońcowym?

Problemy do rozwiązania

10.1. Wiele procesorów umożliwia prowadzenie działań arytmetycznych na upakowanych liczbach dziesiętnych. Chociaż reguły arytmetyki dziesiętnej są podobne do reguł operacji binarnych, wyniki dziesiętne mogą wymagać pewnych poprawek dotyczących pojedynczych cyfr, jeśli jest używana logika binarna.

Rozważmy dziesiętne dodawanie dwóch liczb bezznakowych. Jeśli każda liczba składa się z N cyfr, to w każdej liczbie jest $4N$ bitów. Obie liczby mają być dodane za pomocą sumatora binarnego. Zasugeruj prostą regułę korekty wyniku. Przeprowadź tym sposobem dodawanie liczb 1698 i 1786.

10.2. Uzupełnienie do 10 liczby dziesiętnej X jest definiowane jako $10^N - X$, gdzie N jest liczbą cyfr dziesiętnych w tej liczbie. Opisz użycie reprezentacji uzupełnienia do 10 w celu przeprowadzenia odejmowania dziesiętnego. Zilustruj tę procedurę odejmując $(0326)_{10}$ od $(0736)_{10}$.

10.3. Porównaj maszyny 0, 1, 2 i 3-adresowe, pisząc programy do obliczenia

$$X = (A + B \times C) / (D - E \times F)$$

dla każdej z czterech maszyn. Dostępne są rozkazy:

0-adresowa	1-adresowa	2-adresowa	3-adresowa
PUSH M POP M ADD SUB MUL DIV	LOAD M STORE M ADD M SUB M MUL M DIV M	MOV (X←Y) ADD (X←X + Y) SUB (X←X − Y) MUL (X←X × Y) DIV(X←X/Y)	MOV (X←Y) ADD(X←Y + Z) SUB (X←Y − Z) MUL (X←Y × Z) DIV (X←Y/Z)

10.4. Rozważmy hipotetyczny komputer z listą rozkazów obejmującą tylko dwa n-bitowe rozkazy. Pierwszy bit określa kod operacji, a pozostałe określają jedno z 2^{n-1} n-bitowych słów w pamięci głównej. Dwoma rozkazami są:

SUBS X Odejmij zawartość lokacji X od akumulatora i przechowaj wynik w lokacji X oraz w akumulatorze.

JUMP X Umieść adres X w liczniku programu.

Słowo w pamięci głównej może zawierać albo rozkaz, albo liczbę binarną w notacji uzupełnienia do dwóch. Zademonstruj, że ten repertuar rozkazów jest rozsądnie wystarczający, pokazując, jak mogą być zaprogramowane następujące operacje:

(a) transfer danych: lokacja X do akumulatora, akumulator do lokacji X;
(b) dodawanie: dodaj zawartość lokacji X do akumulatora;
(c) rozgałęzienie warunkowe;
(d) logiczne OR;
(e) operacje wejścia-wyjścia.

10.5. Wiele list rozkazów zawiera rozkaz NOOP oznaczający brak operacji, który nie ma innego wpływu na procesor poza inkrementacją stanu licznika programu. Podaj pewne zastosowania tego rozkazu.

10.6. W podrozdziale 10.4 było powiedziane, że arytmetyczne i logiczne przesunięcia w lewo odpowiadają mnożeniu przez 2, jeśli nie ma przepełnienia; jeśli zaś przepełnienie występuje, operacje przesunięć arytmetycznych i logicznych prowadzą do odmiennych

wyników, jednak przesunięcie arytmetyczne zachowuje znak liczby. Zademonstruj prawdziwość tych stwierdzeń dla 5-bitowych liczb całkowitych w notacji uzupełnienia do dwóch.

10.7. W jaki sposób zaokrągla się liczby przy użyciu arytmetycznego przesunięcia w lewo (np. zaokrąglenie w kierunku +∞, w kierunku −∞, w kierunku zera i w kierunku przeciwnym od zera)?

10.8. Załóżmy, że do zarządzania wywołaniami i powrotami podprogramów standardowych przez procesor ma być użyty stos. Czy można wyeliminować licznik programu przez zastosowanie wierzchołka stosu jako licznika programu?

10.9. W dodatku 10A jest powiedziane, że w liście rozkazów nie występują rozkazy zorientowane na używanie stosu, jeśli stos ma być wykorzystywany tylko przez procesor do takich celów, jak manipulowanie podprogramami. Jak procesor może używać stosu do jakiegokolwiek celu bez rozkazów zorientowanych na używanie stosu?

10.10. Przekształć następujące formuły z odwrotnej notacji polskiej na notację wrostkową (*infix*):

(a) AB + C + D ×
(b) AB/CD/ +
(c) ABCDE + × × /
(d) ABCDE + F/ + G − H/ × +

10.11. Przekształć następujące formuły z notacji wrostkowej na odwrotną polską:

(a) A + B + C + D + E
(b) (A + B) × (C + D) + E
(c) (A × B) + (C × D) + E
(d) (A − B) × (((C − D × E)/F)/G) × H

10.12. Przekształć wyrażenie A + B − C na notację przyrostkową, stosując algorytm Dijkstry. Pokaż kolejne kroki. Czy wynik jest równoważny (A + B) − C lub A + (B − C)? Czy ma to znaczenie?

10.13. Architektura Pentium obejmuje rozkaz nazywany DAA – wyrównanie dziesiętne po dodawaniu (skrót od *decimal adjust after addition*). DAA obejmuje następującą sekwencję rozkazów:

```
if ((AL AND 0FH) > 9) OR (AF = 1) then
     AL ← AL + 6;
     AF ← 1;
else
     AF ← 0;
endif;
if((AL > 9FH) OR (CF = 1) then
     AL← AL + 60H;
     CF← 1;
else
     AF ← 0;
endif.
```

„H" oznacza notację szesnastkową. AL jest rejestrem 8-bitowym, który zachowuje wynik dodawania dwóch bezznakowych 8-bitowych liczb całkowitych. AF jest znacznikiem stanu ustawianym, jeśli w wyniku dodawania wystąpi przeniesienie z bitu 3 do bitu 4. CF jest znacznikiem stanu ustawianym, gdy nastąpi przeniesienie z bitu 7 do 8. Wyjaśnij funkcję realizowaną przez rozkaz DAA.

10.14. Występujący w Pentium rozkaz porównania CMP (*Compare*) powoduje: wykonanie odejmowania argumentu źródłowego od argumentu docelowego, aktualizację znacznika stanu (C, P, A, Z, S, O), jednak nie powoduje zmiany żadnego z argumentów. Po rozkazie CMP może wystąpić rozkaz skoku warunkowego Jcc (*Conditional Jump*) lub zerowania warunkowego SETcc (*Set Condition*), gdzie cc odnosi się do jednego z 16 warunków wymienionych w tabeli 10.10. Zademonstruj, że warunki testowane przy porównywaniu liczby mającej znak są prawidłowe.

10.15. Większość list rozkazów mikroprocesorów zawiera rozkaz, który powoduje sprawdzenie warunku i ustalenie argumentu docelowego, jeśli warunek jest spełniony. Przykładami są SETcc w Pentium, Scc w MC68000 Motoroli oraz Scond w NS32000 firmy National.

(a) Istnieją pewne różnice między tymi rozkazami:

– SETcc i Scc operują tylko na bajtach, podczas gdy Scond operuje na bajtach, słowach i podwójnych słowach.

– SETcc i Scond ustalają argument w postaci liczby całkowitej 1, jeśli warunek jest spełniony, oraz na 0, jeśli nie jest. Scc ustala bajt w postaci wszystkich binarnych jedynek, jeśli warunek jest spełniony, oraz wszystkich zer, gdy nie jest.

Jakie są względne zalety i wady wynikające z tych różnic?

(b) Żaden z tych rozkazów nie ustawia znaczników stanu kodu warunkowego i wobec tego jest wymagane odrębne sprawdzenie wyniku rozkazu w celu określenia jego wartości. Przedyskutuj, czy kody warunkowe powinny być ustawiane jako wynik tego rozkazu.

(c) Prosta instrukcja IF, taka jak IF a > b THEN może być zaimplementowana za pomocą metody reprezentacji numerycznej, to znaczy za pomocą ujawniania wartości Boole'a, w przeciwieństwie do metody *przepływu sterowania* (*flow of control*), w której wartość wyrażenia Boole'a jest reprezentowana przez punkt osiągnięty w programie. Kompilator może wdrożyć IF a > b THEN za pomocą następującego kodu procesora 80×86:

```
    SUB   CX, CX     ;ustaw rejestr CX na 0
    MOV   AX, B      ;przesuń zawartość lokacji B do rejestru AX
    CMP   AX, A      ;porównaj zawartość rejestru AX i lokacji A
    JLE   TEST       ;skocz, jeśli A ≤ B
    INC   CX         ;dodaj 1 do zawartości rejestru CX
    TEST  JCXZ OUT   ;skocz, jeśli zawartość CX równa 0
THEN

OUT
```

Wynik (A > B) jest wartością Boole'a zachowaną w rejestrze i dostępną później poza kontekstem pokazanego wyżej strumienia kodów. Wygodne jest używanie do tego celu rejestru CX, ponieważ wiele kodów operacyjnych skoku i pętli ma wbudowane sprawdzanie rejestru CX.

Pokaż alternatywną implementację, używając rozkazu SETcc, które oszczędzałoby pamięć i czas wykonywania.

(d) Rozważmy teraz instrukcję w języku wysokiego poziomu:

A: = (B > C) OR (D = F)

Kompilator może generować następujący kod:

```
    MOV   EAX, B   ;przenieś zawartość lokacji B do rejestru EAX
    CMP   EAX, C   ;porównaj zawartości rejestru EAX i lokacji C
    MOV   BL, 0    ;0 reprezentuje fałsz (niespełnienie warunku)
    JLE   N1       ;skocz, jeśli B ≤ C
    MOV   BL, 1    ;1 reprezentuje fałsz
N1: MOV   EAX, D
    CMP   EAX, F
    MOV   BH, 0
    JNE   N2
    MOV   BH, 1
N2: OR    BL, BH
```

Pokaż alternatywną implementację z zastosowaniem rozkazu SETcc, które oszczędza pamięć i czas wykonywania.

10.16. Posługując się algorytmem konwersji notacji wrostkowej na przyrostkową przedstawionym w dodatku 10A, pokaż etapy przekształcania wyrażenia z rys. 10.15 na przyrostkowe. Posłuż się prezentacją podobną do pokazanej na rys. 10.17.

10.17. Zademonstruj obliczenie wyrażenia z rys. 10.17, posługując się prezentacją podobną do pokazanej na rys. 10.16.

10.18. Przekształć strukturę typu *najpierw najmłodszy bajt* („cienkokońcową") na rys. 10.18 tak, żeby bajty wyglądały na liczone według schematu *najpierw najstarszy bajt* („grubokońcową"). To znaczy, pokaż pamięć w postaci wierszy 64-bitowych, z bajtami liczonymi od lewej do prawej oraz od góry do dołu.

10.19. Dla następujących struktur danych wykreśl struktury typu *najpierw najstarszy bajt* („grubokońcowe") i *najpierw najmłodszy bajt* („cienkokońcowe"), stosując format z rys. 10.18. Skomentuj wyniki.

(a)
```
struct {
    double i;  //0x1112131415161718
} s1;
```
(b)
```
struct {
    int i;   //0x11121314
    int j;   //0x15161718
} s2;
```
(c)
```
struct {
    short i; //0x1112
    short j; //0x1314
    short k; //0x1516
    short l; //0x1718
} s3;
```

10.20. Specyfikacja architektury PowerPC nie dyktuje, jak powinien być wdrażany tryb *najpierw najmłodszy bajt*. Określa ona tylko widok pamięci, jaki musi postrzegać procesor, pracując w tym trybie. Podczas konwersji struktury danych z *najpierw najstarszy bajt* na *najpierw najmłodszy bajt* istnieje wolność wyboru między implementacją prawdziwego mechanizmu wymiany bajtów lub użyciem pewnego rodzaju mechanizmu modyfikacji adresu. Obecnie wszystkie procesory PowerPC są maszynami pracującymi w trybie *najpierw najstarszy bajt*, stosującymi metodę modyfikacji adresu, aby dane były traktowane jako dane typu *najpierw najmłodszy bajt*.

Rozważmy strukturę *s* zdefiniowaną na rys. 10.18. Prawa dolna część rysunku pokazuje strukturę *s* widzianą przez procesor. W rzeczywistości, jeśli struktura *s* jest utworzona w trybie *najpierw najmłodszy bajt*, jej rozkład w pamięci wygląda tak jak na rys. 10.12.

Odwzorowanie adresu typu *najpierw najmłodszy bajt*

Adres bajta								
					11	12	13	14
00	00	01	02	03	04	05	06	07
	21	22	23	24	25	26	27	28
08	08	09	0A	0B	0C	0D	0E	0F
	'D'	'C'	'B'	'A'	31	32	33	34
10	10	11	12	13	14	15	16	17
			51	52		'G'	'F'	'E'
18	18	19	1A	1B	1C	1D	1E	1F
					61	62	63	64
20	20	21	22	23	24	25	26	27

Rysunek 10.12. Cienkokońcowe struktury w pamięci PowerPC

Wyjaśnij wykorzystane odwzorowanie, opisz prosty sposób wdrożenia odwzorowania i przedyskutuj efektywność tego rozwiązania.

10.21. Napisz niewielki program określający „końcowość" komputera. Uruchom ten program na dostępnym komputerze i przedstaw wynik.

Dodatek 10A. Stosy

Stosy

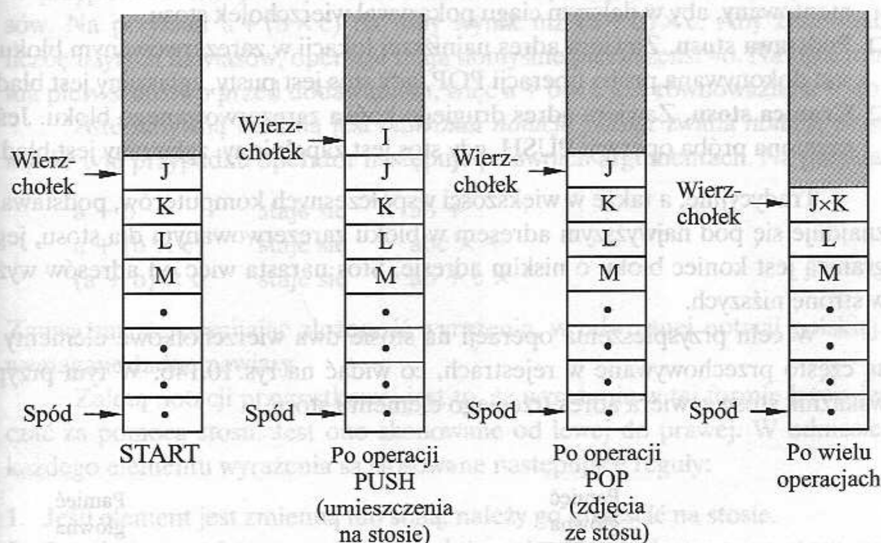
Stos jest uporządkowanym zestawem elementów, z których tylko do jednego można mieć dostęp w określonej chwili. Punkt dostępu nazywa się *wierzchołkiem* stosu. Liczba elementów w stosie, czyli *długość* stosu, jest zmienna. Obiekty mogą być dodawane lub odejmowane tylko z wierzchołka stosu. Z tego powodu stos jest również znany jako lista „spychania" (*pushdown list*) lub *lista typu ostatni na wejściu, pierwszy na wyjściu* (*last-in-first-out list* – LIFO).

Tabela 10.13. Operacje dotyczące stosu

PUSH	Umieszczenie nowego elementu na wierzchołku stosu
POP	Usunięcie wierzchołkowego elementu stosu
Operacja jednoargumentowa	Wykonanie operacji na wierzchołkowym elemencie stosu. Zamiana elementu wierzchołkowego na wynik
Operacja dwuargumentowa	Wykonanie operacji na dwóch wierzchołkowych elementach stosu. Usunięcie dwóch wierzchołkowych elementów stosu i umieszczenie wyniku operacji na wierzchołku stosu

Na rysunku 10.13 są pokazane podstawowe operacje, które mogą być realizowane na stosie. Rozpoczniemy w chwili, w której stos zawiera pewną liczbę elementów. Operacja PUSH to dodanie nowego obiektu na wierzchołek stosu; operacja POP – usunięcie wierzchołkowego elementu ze stosu. W obu przypadkach wierzchołek stosu ulega odpowiedniemu przesunięciu. Ponadto operacje binarne wymagające dwóch

argumentów (np. mnożenie, dzielenie, dodawanie, odejmowanie) używają dwóch wierzchołkowych elementów stosu jako argumentów, po czym wprowadzają wynik na stos. Operacje jednoargumentowe (np. logiczne NOT) używają wierzchołkowego elementu stosu. Wszystkie te operacje są zebrane w tabeli 10.13.



Rysunek 10.13. Podstawowe operacje dotyczące stosu

Implementacja stosu

Stos jest na tyle użyteczną strukturą, że stanowi część implementacji procesora. Jednym z zastosowań, przedyskutowanym w podrozdz. 10.4, jest zarządzanie wywołaniami i powrotami procedur. Stosy mogą też być użyteczne dla programistów. Przykładem tego jest obliczanie wyrażeń, omówione w dalszej części tego punktu.

Implementacja stosu zależy częściowo od jego potencjalnych zastosowań. Jeśli jest pożądane, aby operacje na stosie były dostępne dla programisty, to lista rozkazów będzie zawierała specjalne operacje dotyczące stosu, włącznie z PUSH, POP i operacjami, które używają jednego lub dwóch wierzchołkowych elementów jako argumentów. Ponieważ wszystkie te operacje odnoszą się do jednoznacznej lokacji, mianowicie do wierzchołka stosu, adres argumentu lub argumentów jest domyślny i nie musi być włączany do rozkazu. Są to więc rozkazy 0-adresowe omówione w podrozdz. 10.1.

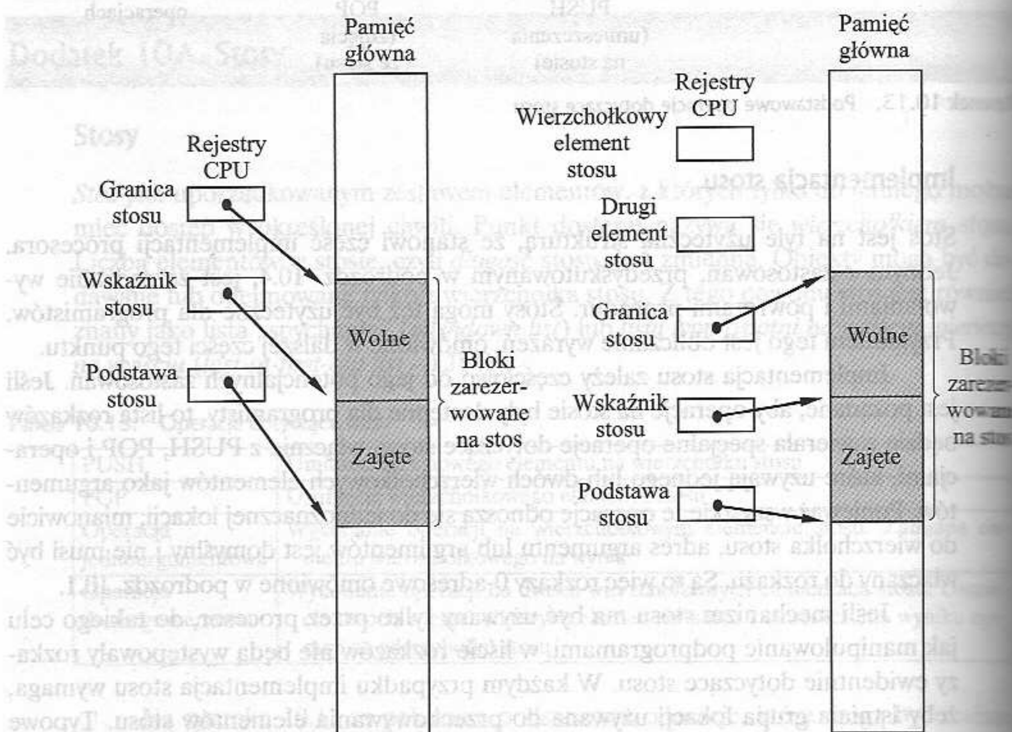
Jeśli mechanizm stosu ma być używany tylko przez procesor, do takiego celu jak manipulowanie podprogramami, w liście rozkazów nie będą występowały rozkazy ewidentnie dotyczące stosu. W każdym przypadku implementacja stosu wymaga, żeby istniała grupa lokacji używana do przechowywania elementów stosu. Typowe rozwiązanie zilustrowano na rys. 10.14a. W pamięci głównej lub w wirtualnej jest zarezerwowany blok sąsiadujących lokacji przeznaczony na stos. Przez większość czasu blok jest częściowo wypełniony elementami stosu, pozostałość zaś jest przewidziana na rozrastanie się stosu.

Do poprawnego działania są wymagane trzy adresy; często są one przechowywane w rejestrach procesora:

- **Wskaźnik stosu.** Zawiera adres wierzchołka stosu. Jeśli element jest dodawany lub usuwany ze stosu, wskaźnik jest odpowiednio inkrementowany lub dekrementowany, aby w dalszym ciągu pokazywał wierzchołek stosu.
- **Podstawa stosu.** Zawiera adres najniższej lokacji w zarezerwowanym bloku. Jeśli jest dokonywana próba operacji POP, gdy stos jest pusty, zgłaszany jest błąd.
- **Granica stosu.** Zawiera adres drugiego końca zarezerwowanego bloku. Jeśli jest czyniona próba operacji PUSH, gdy stos jest zapełniony, zgłaszany jest błąd.

Tradycyjnie, a także w większości współczesnych komputerów, podstawa stosu znajduje się pod najwyższym adresem w bloku zarezerwowanym dla stosu, jego zaś granicą jest koniec bloku o niskim adresie. Stos narasta więc od adresów wyższych w stronę niższych.

W celu przyspieszenia operacji na stosie dwa wierzchołkowe elementy stosu są często przechowywane w rejestrach, co widać na rys. 10.14b. W tym przypadku wskaźnik stosu zawiera adres trzeciego elementu stosu.



Rysunek 10.14. Typowe organizacje stosu

Obliczanie wyrażeń

Wzory matematyczne są zwykle wyrażane za pomocą notacji nazywanej *wrostkową* (*infix*). W tej postaci operator binarny występuje między argumentami (np. a + b). W przypadku złożonych wyrażeń do określania porządku obliczeń używa się nawiasów. Na przykład a + (b × c) da inny wynik niż (a + b) × c. Aby zminimalizować liczbę użytych nawiasów, operacje mają domyślne pierwszeństwo. Na ogół mnożenie ma pierwszeństwo przed dodawaniem, więc a + b × c jest równoważne a + (b × c).

Alternatywną metodą jest *odwrotna notacja polska* zwana *notacją przyrostkową*. W tym przypadku operator następuje po swoich argumentach. Na przykład

a + b	staje się	ab +
a + (b × c)	staje się	abc × +
(a + b) × c	staje się	ab + c ×

Zauważmy, że pomijając złożoność wyrażenia, w odwrotnej notacji polskiej nie są wymagane żadne nawiasy.

Zaletą notacji przyrostkowej jest to, że wyrażenie w tej formie łatwo jest obliczać za pomocą stosu. Jest ono skanowane od lewej do prawej. W odniesieniu do każdego elementu wyrażenia są stosowane następujące reguły:

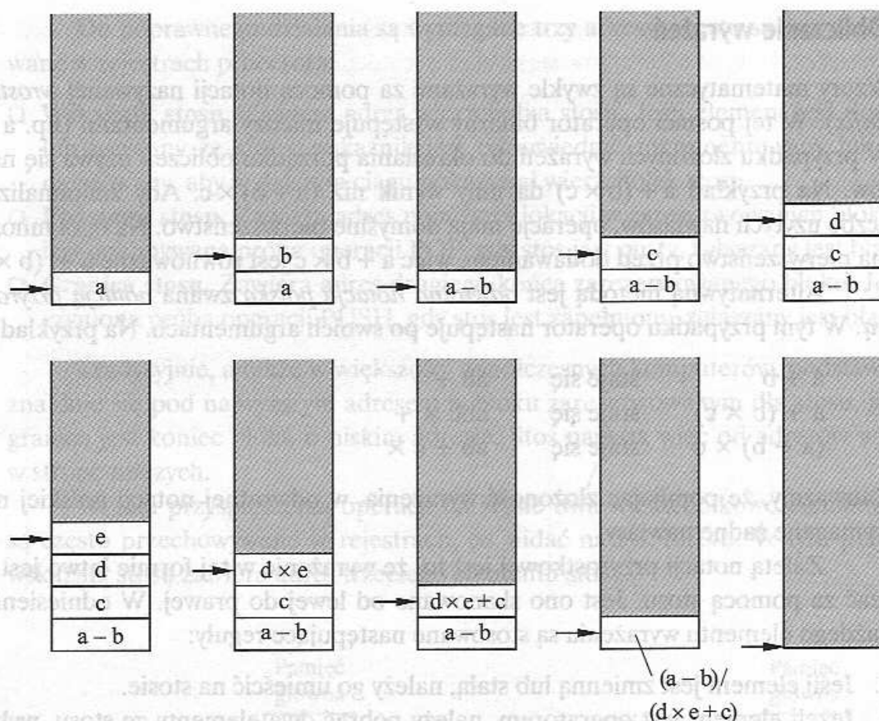
1. Jeśli element jest zmienną lub stałą, należy go umieścić na stosie.
2. Jeżeli element jest operatorem, należy pobrać dwa elementy ze stosu, wykonać operację i umieścić wynik na stosie.

Po przejrzeniu całego wyrażenia wynik znajduje się na wierzchołku stosu.

	Stos	Rejestry robocze	Pojedynczy rejestr
	Umieść a Umieść b Odejmij Umieść c Umieść d Umieść e Pomnóż Dodaj Podziel Zdejmij f	Ładuj Rl, a Odejmij G1, b Ładuj R2, d Pomnóż R2, e Dodaj R2, c Podziel R1, R2 Zapisz R1, f	Ładuj d Pomnóż e Dodaj c Zapisz f Ładuj a Odejmij b Podziel f Zapisz f
Liczba rozkazów	10	7	8
Dostęp do pamięci	10op+6d	7op+6d	8op+8d

Rysunek 10.15. Porównanie trzech programów do obliczania f = (a − b) / (c + d × e)

Prostota tego algorytmu powoduje, że jest on wygodny przy obliczaniu wyrażeń. Dlatego też wiele kompilatorów dokonuje konwersji wyrażeń w języku wysokiego poziomu na notację przyrostkową, a następnie generuje rozkazy maszynowe odnoszące się do tej notacji. Na rysunku 10.15 widać sekwencję rozkazów maszynowych służącą do obliczenia f = (a − b) / (c + d × e) za pomocą rozkazów wykorzystujących



Rysunek 10.16. Użycie stosu do obliczenia f = (a – b) / (d × e + c)

stos. Jest także pokazane użycie rozkazów 1- i 2-adresowych. Reguły stosu nie były używane w ostatnich dwóch przypadkach; zauważmy, że mimo to notacja przyrostkowa służyła jako przewodnik do generowania rozkazów maszynowych. Sekwencję zdarzeń dla programu wykorzystującego stos widać na rys. 10.16.

Proces konwersji wyrażenia wrostkowego na przyrostkowe może być najłatwiej zrealizowany również za pomocą stosu. Następujący algorytm zawdzięczamy Dijkstrze [DIJK63]. Wyrażenie wrostkowe jest przeglądane od lewej do prawej, a wyrażenie przyrostkowe jest tworzone i wyprowadzane podczas tego przeglądania. Etapy tego procesu są następujące:

1. Zbadaj następny element na wejściu.
2. Jeśli jest on argumentem, skieruj go do wyjścia.
3. Jeśli jest on nawiasem otwierającym, umieść go na stosie.
4. Jeśli jest on operatorem, to

- jeśli na wierzchołku stosu znajduje się nawias otwierający, to umieść operator na stosie;
- jeśli ma on wyższy priorytet niż wierzchołek stosu (mnożenie i dzielenie mają wyższy priorytet niż dodawanie i odejmowanie), to umieść operator na stosie;
- w innym przypadku skieruj operację ze stosu do wyjścia i powtórz krok 4.

5. Jeśli jest on nawiasem zamykającym, to pobieraj operatory do wyjścia, aż napotkasz nawias otwierający. Pobierz i wyrzuć nawias otwierający.
6. Jeśli istnieją jeszcze dane na wejściu, przejdź do kroku 1.
7. Jeśli nie ma danych na wejściu, pobierz ze stosu pozostałe argumenty.

Rysunek 10.17 ilustruje użycie tego algorytmu. Przykład ten powinien dać czytelnikowi pewne odczucie potęgi algorytmów opartych na stosach.

Wejście	Wyjście	Stos (wierzchołek po prawej)
A + B × C + (D + E) × F	pusty	pusty
+ B × C + (D + E) × F	A	pusty
B × C + (D + E) × F	A	+
× C + (D + E) × F	AB	+
C + (D + E) × F	AB	+ ×
+ (D + E) × F	ABC	+ ×
(D + E) × F	ABC × +	+
D + E) × F	ABC × +	+(
+ E) × F	ABC × + D	+(
E) × F	ABC × + D	+(+
) × F	ABC × + DE	+(+
× F	ABC × + DE +	+
F	ABC × + DE +	+ ×
pusty	ABC × + DE+F	+ ×
pusty	ABC × +DE + F × +	pusty

Rysunek 10.17. Konwersja wyrażenia z notacji wrostkowej na przyrostkową

Dodatek 10B. Struktury „cienko-", „grubo-" i „dwukońcowe"

Zjawisko (przykre i osobliwe), o którym będziemy dyskutować, wiąże się ze sposobem reprezentowania i odnoszenia się do bajtów wewnątrz słowa oraz do bitów wewnątrz bajta. Najpierw rozważymy problem porządkowania bajtów, następnie zaś bitów.

Porządkowanie bajtów

Koncepcja „końcowości" była po raz pierwszy przedyskutowana w literaturze przez Cohena [COHE81]. W odniesieniu do bajtów dotyczy to porządkowania bajtów w wielobajtowych wartościach skalarnych. Najlepiej jest przedstawić to zagadnienie za pomocą przykładu. Załóżmy, że mamy 32-bitową wartość w notacji szesnastkowej 12345678 i że jest ona przechowywana w 32-bitowym słowie w pamięci umożliwiającej adresowanie bajtowe, w lokacji bajtowej 184. Wartość składa się z 4 bajtów, przy

czym najmniej znaczący bajt zawiera wartość 78, zaś najbardziej znaczący bajt zawiera wartość 12. Istnieją dwa sposoby przechowywania tej wartości:

Adres	Wartość		Adres	Wartość
184	12		184	78
185	34		185	56
186	56		186	34
187	78		187	12

Odwzorowanie przedstawione po lewej stronie polega na przechowywaniu najbardziej znaczącego bajta pod najniższym numerycznym adresem bajtowym; jest to równoważne kierunkowi pisania od lewej do prawej przyjętemu w kulturze zachodniej. Takie odwzorowanie jest określane jako „grubokońcowe". Odwzorowanie po prawej przewiduje umieszczenie najmniej znaczącego bajta pod najniższym numerycznym adresem bajtowym; jest ono znane jako „cienkokońcowe" i jest reminiscencją porządku od prawej do lewej operacji arytmetycznych w jednostkach arytmetycznych[2*]. „Grubokońcowa" struktura wielobajtowej wielkości skalarnej jest odwróconym odwzorowaniem bajtowym struktury „cienkokońcowej".

Problem końcowości powstaje, gdy jest konieczne traktowanie wielobajtowej wielkości jako jednostki danych o jednym adresie, jeśli nawet składa się ona z mniejszych, adresowalnych elementów. Niektóre maszyny, takie jak Intel 80x86, Pentium, VAX i Alpha są maszynami „cienkokońcowymi", podczas gdy inne, takie jak IBM System 370/390, Motorola 680x0, Sun SPARC i większość maszyn RISC są „grubokońcowe". Wynikają stąd problemy, gdy dane są przenoszone z maszyny jednego rodzaju do maszyny drugiego rodzaju i gdy programista chce manipulować indywidualnymi bajtami lub bitami wewnątrz wielobajtowej wielkości skalarnej.

Własność końcowości nie rozciąga się poza indywidualną jednostkę danych. W dowolnej maszynie zbiory, takie jak pliki, struktury danych i matryce są złożone z wielu jednostek danych, z których każda ma określoną „końcowość". Wobec tego konwersja bloku pamięci z jednego stylu „końcowości" do drugiego wymaga znajomości struktury danych.

Na rysunku 10.18 jest pokazane, jak końcowość determinuje adresowanie i porządek bajtów. Struktura C w górnej części rysunku zawiera pewną liczbę rodzajów danych. Rozplanowanie pamięci pokazane w lewej dolnej części rysunku wynika z ułożenia tej struktury pod kątem maszyny „grubokońcowej", a pokazane w prawej dolnej części – pod kątem maszyny „cienkokońcowej". W każdym przypadku pamięć jest przedstawiona w postaci ciągu wierszy 64-bitowych. W przypadku „grubo-

---

[2] Wyrażenia „grubokońcowy" i „cienkokońcowy" pochodzą z rozdz. 4 części I „Podróży Guliwera do wielu odległych narodów świata" Jonathana Swifta. Odnoszą się do wojny religijnej między dwiema grupami (Grubokońców i Cienkokońców), jedną tłukącą jajka od grubszego końca i drugą tłukącą je od cieńszego końca.

[*] Wydanie polskie w przekładzie Macieja Słomczyńskiego, Wydawnictwo Literackie, Kraków 197[illegible] (*przyp. tłum.*).

końcowym" pamięć jest układana od lewej do prawej i od góry do dołu, podczas gdy w przypadku „cienkokońcowym" – od prawej do lewej i od dołu do góry. Zauważmy, że te rozplanowania są arbitralne. Każdy z tych schematów mógłby używać wewnątrz wiersza kierunku od lewej do prawej lub od prawej do lewej; jest to problem zobrazowania, a nie przyporządkowania pamięci. W rzeczywistości, przeglądając podręczniki programowania dla różnych maszyn, obserwujemy zadziwiającą kolekcję zobrazowań, nawet wewnątrz tego samego podręcznika.

```
Struct{
int       a;        / /0x1112_1314                   Słowo
int       pad;
double    b;        / /0x2122_2324_2526_2728         Podwójne słowo
char*     c;        / /0x3132_3334                   Słowo
char      d [7];    / /'A','B','C','D','E','F','G'   Tablica bajtów
short     e;        / /0x5152                        Półsłowo
int       f;        / / 0x6162_6364                  Słowo
```

Odwzorowanie adresu typu *najpierw najstarszy bajt*

Adres bajta								
	11	**12**	**13**	**14**				
00	00	01	02	03	04	05	06	07
	21	**22**	**23**	**24**	**25**	**26**	**27**	**28**
08	08	09	0A	0B	0C	0D	0E	0F
	31	**32**	**33**	**34**	**'A'**	**'B'**	**'C'**	**'D'**
10	10	11	12	13	14	15	16	17
	'E'	**'F'**	**'G'**		**51**	**52**		
18	18	19	1A	1B	1C	1D	1E	1F
	61	**62**	**63**	**64**				
20	20	21	22	23				

Odwzorowanie adresu typu *najpierw najmłodszy bajt*

								Adres bajta
				11	**12**	**13**	**14**	
07	06	05	04	03	02	01	00	00
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	
0F	0E	0D	0C	0B	0A	09	08	08
'D'	**'C'**	**'B'**	**'A'**	**31**	**32**	**33**	**34**	
17	16	15	14	13	12	11	10	10
		51	**52**	**'G'**	**'F'**	**'E'**		
1F	1E	1D	1C	1B	1A	19	18	18
				61	**62**	**63**	**64**	
				23	22	21	20	20

Rysunek 10.18. Przykład struktury danych C oraz jej odwzorowania „grubo-" i „cienkokońcowe"

Możemy poczynić kilka spostrzeżeń w odniesieniu do tej struktury danych:

- Każdy element danych ma taki sam adres w obu schematach. Na przykład adresem podwójnego słowa o wartości szesnastkowej 2122232425262728 jest 08.
- Wewnątrz dowolnej wielobajtowej wartości skalarnej uporządkowanie bajtów w strukturze „cienkokońcowej" jest odwrotne do uporządkowania w strukturze „grubokońcowej".
- Końcowość nie wpływa na uporządkowanie elementów danych wewnątrz struktury. Wobec tego 4-znakowe słowo c ma odwrócony porządek bajtów, a 7-znakowy zespół bajtów d nie. W związku z tym adres każdego indywidualnego elementu d jest taki sam w obu strukturach.

Wpływ końcowości można jaśniej zademonstrować, spoglądając na pamięć jako na pionowy zespół bajtów, co widać na rys. 10.19.

(a) adres	(a) zawartość	(b) adres	(b) zawartość
00	11	00	14
	12		13
	13		12
	14		11
04		04	
08	21	08	28
	22		27
	23		26
	24		25
0C	25	0C	24
	26		23
	27		22
	28		21
10	31	10	34
	32		33
	33		32
	34		31
14	'A'	14	'A'
	'B'		'B'
	'C'		'C'
	'D'		'D'
18	'E'	18	'E'
	'F'		'F'
	'G'		'G'
1C	51	1C	52
	52		51
20	61	20	64
	62		63
	63		62
	64		61

(a) „grubokońcowe" (b) „cienkokońcowe"

Rysunek 10.19. Inne ujęcie rys. 10.18

Nie istnieje zgodny pogląd co do tego, który styl końcowości jest lepszy[3]. O
co przemawia za stylem „grubokońcowym":

- **Sortowanie łańcuchów znaków.** Procesor „grubokońcowy" szybciej porównuj wyrównane łańcuchy znaków; ALU może równolegle porównywać wiele ba- tów.

---

[3] Prorok szanowany przez obie grupy w Wojnach Końcowych w „Podróżach Guliwera do wie odległych narodów świata" miał do powiedzenia, co następuje: „Niechaj wszyscy wierni tłuką jaja z ka dziej dogodnego końca". Niezbyt pomocne!

- **Zrzuty dziesiętne/IRA.** Wszystkie wartości mogą być drukowane od lewej do prawej bez wywoływania nieporozumień.
- **Spójny porządek.** Procesory „grubokońcowe" przechowują łańcuchy liczb całkowitych i znaków w tym samym porządku (jako pierwszy bajt najbardziej znaczący).

Następujące punkty natomiast przemawiają za stylem „cienkokońcowym":

- Procesor „grubokońcowy" musi wykonać dodawanie, gdy przekształca adres 32-bitowej liczby całkowitej na adres 16-bitowej liczby całkowitej w celu użycia najmniej znaczących bajtów.
- W stylu „cienkokońcowym" łatwiej jest wykonywać działania arytmetyczne z dużą dokładnością; nie musi się szukać najmniej znaczącego bajta i wracać.

Różnice są niewielkie i wybór stylu końcowości jest często raczej sprawą przystosowania poprzednich maszyn niż czegokolwiek innego.

PowerPC jest procesorem „dwukońcowym" (*bi-endian*), gdyż może pracować zarówno w trybie „grubokońcowym", jak i „cienkokońcowym". Architektura „dwukońcowa" umożliwia programistom wybieranie dowolnego trybu, gdy przenoszony jest system operacyjny i programy użytkowe z innych maszyn. System operacyjny ustala tryb końcowości, w którym realizuje procesy. Gdy już tryb został wybrany, wszystkie następne załadowania pamięci są wyznaczone przez model adresowania wynikający z tego trybu. Aby umożliwić obsługę tej cechy sprzętowej, pozostawione zostały dwa bity w rejestrze stanu maszyny (MSR). Jeden bit służy do określenia trybu końcowości, w jakim pracuje jądro; drugi – do określenia bieżącego trybu pracy procesora. Wobec tego tryb może być zmieniany od procesu do procesu.

Porządkowanie bitów

Porządkując bity wewnątrz bajta, natychmiast stykamy się z dwiema kwestiami:

1. Czy liczymy pierwszy bit jako bit 0, czy jako bit 1?
2. Czy przypisujemy najniższy numer bitowy najmniej znaczącemu bitowi bajta (styl „cienkokońcowy"), czy też najbardziej znaczącemu bitowi bajta (styl „grubokońcowy")?

Odpowiedzi na te pytania nie są jednakowe w przypadku wszystkich maszyn. W rzeczywistości w przypadku niektórych maszyn odpowiedzi są różne w różnych okolicznościach. Ponadto wybór „grubokońcowego" lub „cienkokońcowego" porządku bitów w bajcie nie zawsze jest spójny z uporządkowaniem bajtów wewnątrz wielobajtowego skalara. Programista musi interesować się tymi zagadnieniami, manipulując indywidualnymi bitami.

Innym obszarem zainteresowań jest transmitowanie danych bitową linią szeregową. Gdy jest transmitowany pojedynczy bajt, czy system transmituje najpierw bit najbardziej znaczący, czy najmniej znaczący? Projektant musi się upewnić, że nadchodzące bity są właściwie interpretowane i traktowane. Omówienie tego zagadnienia znajduje się w [JAME90].

(a) „grubokońcowe"

(b) „cienkokońcowe"

Rozdział 11

Listy rozkazów: tryby adresowania i formaty rozkazów

11.1. Adresowanie

Adresowanie natychmiastowe
Adresowanie bezpośrednie
Adresowanie pośrednie
Adresowanie rejestrowe
Adresowanie rejestrowe pośrednie
Adresowanie z przesunięciem
Adresowanie stosowe

11.2. Tryby adresowania Pentium i PowerPC

Tryby adresowania Pentium
Tryby adresowania PowerPC

11.3. Formaty rozkazów

Długość rozkazu
Przydział bitów
Rozkazy o zmiennej długości

11.4. Formaty rozkazów Pentium i PowerPC

Formaty rozkazów Pentium
Formaty rozkazów PowerPC

11.5. Polecana literatura

11.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

PODSTAWOWE SPOSTRZEŻENIA

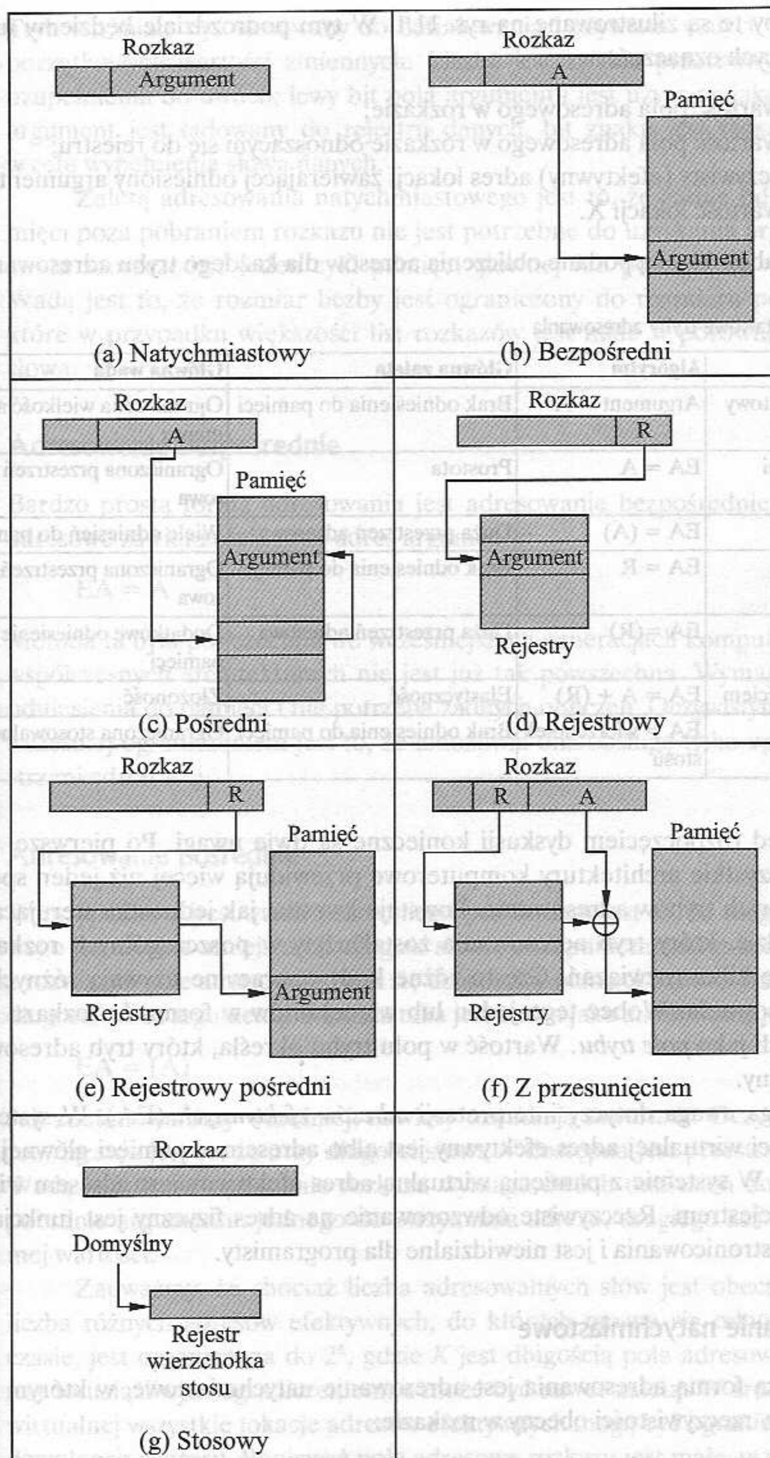
- Odniesienie do argumentu w rozkazie zawiera albo rzeczywistą wartość argumentu (odniesienie natychmiastowe), albo odniesienie do adresu tego argumentu. W różnych listach rozkazów stosuje się różne tryby adresowania. Należą do nich: adresowanie bezpośrednie (adres argumentu znajduje się w polu adresu), pośrednie (pole adresu wskazuje na lokację zawierającą adres argumentu), rejestrowe, rejestrowe pośrednie oraz różne formy przesunięć, w przypadku których wartość rejestrowa jest dodawana do wartości adresowej w celu utworzenia adresu argumentu.
- Format rozkazu określa układ pól w rozkazie. Projektowanie formatu rozkazu jest przedsięwzięciem złożonym, w którym muszą być uwzględnione takie rozważania, jak długość rozkazu, jej stałość lub zmienność, liczba bitów przypisanych do kodu operacji oraz do każdego odniesienia do argumentu, a także sposób określania trybu adresowania.

W rozdziale 10 skupiliśmy się na tym, jaką rolę pełni lista rozkazów. W szczególności przeanalizowaliśmy rodzaje argumentów i operacji, które mogą być określane za pomocą rozkazów maszynowych. W tym rozdziale zajmiemy się zagadnieniem, jak określać argumenty i operacje w rozkazie. Występują tu dwa problemy. Po pierwsze, jak jest określany adres argumentu, i po drugie, jak są organizowane bity rozkazu w celu określenia adresów argumentów i operacji.

11.1. Adresowanie

Pole lub pola adresowe w typowym formacie rozkazu są stosunkowo niewielkie. Chcielibyśmy dysponować możliwością odnoszenia się do dużego zakresu lokacji w pamięci głównej lub – w pewnych systemach – wirtualnej. Aby osiągnąć ten cel, zastosowano różnorodne metody adresowania. Wszystkie te metody opierają się na kompromisie między zakresem adresów i (lub) elastycznością adresowania z jednej strony, a liczbą odniesień do pamięci i (lub) złożonością obliczania adresów z drugiej strony. W tym podrozdziale przeanalizujemy najpowszechniejsze tryby adresowania:

- natychmiastowy;
- bezpośredni;
- pośredni;
- rejestrowy;
- rejestrowy pośredni;
- z przesunięciem;
- stosowy.



Rysunek 11.1. Tryby adresowania

Tryby te są zilustrowane na rys. 11.1. W tym podrozdziale będziemy używać następujących oznaczeń:

A – zawartość pola adresowego w rozkazie;
R – zawartość pola adresowego w rozkazie odnoszącym się do rejestru;
EA – rzeczywisty (efektywny) adres lokacji zawierającej odniesiony argument;
(X) – zawartość lokacji X.

W tabeli 11.1 są podane obliczenia adresów dla każdego trybu adresowania.

Tabela 11.1. Podstawowe tryby adresowania

Tryb	Algorytm	Główna zaleta	Główna wada
Natychmiastowy	Argument = A	Brak odniesienia do pamięci	Ograniczona wielkość argumentu
Bezpośredni	EA = A	Prostota	Ograniczona przestrzeń adresowa
Pośredni	EA = (A)	Duża przestrzeń adresowa	Wiele odniesień do pamięci
Rejestrowy	EA = R	Brak odniesienia do pamięci	Ograniczona przestrzeń adresowa
Rejestrowy pośredni	EA =(R)	Duża przestrzeń adresowa	Dodatkowe odniesienie do pamięci
Z przesunięciem	EA = A + (R)	Elastyczność	Złożoność
Stosowe	EA = wierzchołek stosu	Brak odniesienia do pamięci	Ograniczona stosowalność

Przed rozpoczęciem dyskusji konieczne są dwie uwagi. Po pierwsze, praktycznie wszystkie architektury komputerowe przewidują więcej niż jeden spośród wymienionych trybów adresowania. Powstaje kwestia, jak jednostka sterująca może stwierdzić, który tryb adresowania został użyty w poszczególnych rozkazach. Stosuje się kilka rozwiązań. Często różne kody operacyjne używają różnych trybów adresowania. Wobec tego jeden lub więcej bitów w formacie rozkazu może być użytych jako *pole trybu*. Wartość w polu trybu określa, który tryb adresowania jest używany.

Druga uwaga dotyczy interpretacji *adresów efektywnych* (EA). W systemach bez pamięci wirtualnej adres efektywny jest albo adresem w pamięci głównej, albo rejestrem. W systemie z pamięcią wirtualną adres efektywny jest adresem wirtualnym lub rejestrem. Rzeczywiste odwzorowanie na adres fizyczny jest funkcją mechanizmu stronicowania i jest niewidzialne dla programisty.

Adresowanie natychmiastowe

Najprostszą formą adresowania jest adresowanie natychmiastowe, w którym argument jest w rzeczywistości obecny w rozkazie:

ARGUMENT = A

Tryb ten może być stosowany do definiowania i używania stałych lub do ustalania początkowych wartości zmiennych. Liczba jest zwykle przechowywana w postaci uzupełnienia do dwóch; lewy bit pola argumentu jest używany jako bit znaku. Gdy argument jest ładowany do rejestru danych, bit znaku jest rozszerzany na lewo w celu wypełnienia słowa danych.

Zaletą adresowania natychmiastowego jest to, że żadne odniesienie do pamięci poza pobraniem rozkazu nie jest potrzebne do uzyskania argumentu, co pozwala zaoszczędzić jeden cykl pamięci głównej lub podręcznej w cyklu rozkazu. Wadą jest to, że rozmiar liczby jest ograniczony do rozmiaru pola adresowego, które w przypadku większości list rozkazów jest małe w porównaniu z długością słowa.

Adresowanie bezpośrednie

Bardzo prostą formą adresowania jest adresowanie bezpośrednie, w którym pole adresowe zawiera efektywny adres argumentu:

$$EA = A$$

Metoda ta była powszechna we wcześniejszych generacjach komputerów, jednak we współczesnych architekturach nie jest już tak powszechna. Wymaga tylko jednego odniesienia do pamięci i nie potrzeba żadnych obliczeń. Oczywistym, wspomnianym wcześniej ograniczeniem jest to, że umożliwia ona obsługę tylko ograniczonej przestrzeni adresowej.

Adresowanie pośrednie

W przypadku adresowania pośredniego problemem jest to, że długość pola adresowego jest zwykle mniejsza niż długość słowa, co ogranicza zakres adresów. W jednym z rozwiązań pole adresowe odnosi się do słowa w pamięci, które z kolei zawiera pełnej długości adres argumentu. Metoda taka jest znana jako *adresowanie pośrednie*:

$$EA = (A)$$

Jak zdefiniowaliśmy wcześniej, nawiasy oznaczają *zawartość*. Oczywistą zaletą tego rozwiązania jest to, że przy długości słowa N dostępna jest przestrzeń adresowa 2^N. Wadą jest to, że wykonanie rozkazu wymaga dwóch odniesień do pamięci w celu pobrania argumentu: jednego do otrzymania adresu, drugiego zaś do uzyskania samej wartości.

Zauważmy, że chociaż liczba adresowalnych słów jest obecnie równa 2^N, to liczba różnych adresów efektywnych, do których można się odnosić w dowolnym czasie, jest ograniczona do 2^K, gdzie K jest długością pola adresowego. Zwykle nie jest to uciążliwym ograniczeniem, a może być nawet zaletą. W środowisku pamięci wirtualnej wszystkie lokacje adresów efektywnych mogą być ograniczone do strony 0 dowolnego procesu. Ponieważ pole adresowe rozkazu jest małe, w naturalny sposób będzie określało adresy bezpośrednie o niskich numerach, które będą dotyczyły właś-

nie strony 0 (jedynym ograniczeniem jest to, że rozmiar strony musi być większy lub równy 2^K). Gdy proces jest aktywny, powtarzają się odniesienia do strony 0, co powoduje pozostawanie jej w pamięci rzeczywistej. Wobec tego pośrednie odniesienie do pamięci może zawierać co najwyżej jeden błąd strony, a nie dwa.

Rzadko stosowanym wariantem adresowania pośredniego jest *adresowanie wielopoziomowe* lub *kaskadowe*:

EA = (... (A) ...)

W tym przypadku jeden bit adresu obejmującego całe słowo jest znacznikiem pośredniości (I). Jeśli bit I jest 0, to słowo zawiera EA. Jeśli bit I jest 1, to wywoływany jest następny poziom pośredniości. Nie wygląda na to, żeby to rozwiązanie przyniosło szczególne korzyści, natomiast jego wadą jest to, że pobranie argumentu wymaga trzech lub więcej odniesień do pamięci.

Adresowanie rejestrowe

Adresowanie rejestrowe jest podobne do bezpośredniego. Jedyną różnicą jest to, że pole adresowe odnosi się do rejestru zamiast do adresu w pamięci głównej:

EA = R

Pole adresowe odnoszące się do rejestrów ma zwykle 3÷5 bitów, wobec tego można się odwoływać do 8÷32 rejestrów roboczych.

Zaletą adresowania rejestrowego jest to, że (1) w rozkazie jest potrzebne tylko niewielkie pole adresowe, oraz (2) nie są wymagane odniesienia do pamięci. Jak stwierdziliśmy w rozdz. 4, czas dostępu do wewnętrznego rejestru procesora jest znacznie mniejszy niż czas dostępu do pamięci głównej. Wadą adresowania rejestrowego jest bardzo ograniczona przestrzeń adresowa.

Jeśli w liście rozkazów jest głównie używane adresowanie rejestrowe, oznacza to intensywne korzystanie z rejestrów procesora. Ze względu na poważnie ograniczoną liczbę rejestrów (w porównaniu z lokacjami w pamięci głównej), takie korzystanie ma sens tylko wtedy, kiedy jest ono efektywne. Jeśli każdy argument jest doprowadzany do rejestru z pamięci głównej, dokonuje się na nim operacji, po czym jest on zwracany do pamięci głównej, to w ten sposób wprowadzamy dodatkowy, czasochłonny krok. Jeśli natomiast argument pozostaje w rejestrze i jest używany w wielu operacjach, osiągana jest realna oszczędność. Przykładem jest pośredni wynik obliczeń. W szczególności załóżmy, że jest wdrażany w postaci programu algorytm mnożenia w notacji uzupełnienia do dwóch. Do lokacji oznaczonej przez A w sieci działań (patrz rys. 9.12) następuje tak dużo odniesień, że powinna ona być wdrożona raczej w rejestrze niż jako lokacja w pamięci głównej.

Do programisty należy decyzja, które wartości powinny pozostawać w rejestrach, a które powinny być przechowywane w pamięci głównej. Większość nowoczesnych procesorów zawiera wiele rejestrów ogólnego przeznaczenia, co ma na celu usprawnienie pracy programisty posługującego się językiem asemblerowym (np. piszącego kompilatory).

Adresowanie rejestrowe pośrednie

Podobnie jak adresowanie rejestrowe jest analogiczne do adresowania bezpośredniego, tak adresowanie rejestrowe pośrednie jest analogiczne do adresowania pośredniego. W obu przypadkach jedyną różnicą jest to, czy pole adresowe odnosi się do lokacji w pamięci, czy do rejestru. Wobec tego w przypadku pośredniego adresu rejestrowego

$$EA = (R)$$

Zalety i ograniczenia pośredniego adresowania rejestrowego są w zasadzie takie same jak adresowania pośredniego. W obu przypadkach ograniczenie przestrzeni adresowej (ograniczony zakres adresów) pola adresowego jest eliminowane w ten sposób, że pole to odnosi się do zawierającej adres lokacji o długości słowa. Ponadto, stosując pośrednie adresowanie rejestrowe, używa się o jedno odniesienie do pamięci mniej niż przy adresowaniu pośrednim.

Adresowanie z przesunięciem

Bardzo efektywny tryb adresowania łączy możliwości adresowania bezpośredniego oraz pośredniego adresowania rejestrowego. Jest on znany pod wieloma nazwami zależnie od zastosowania, jednak podstawowy mechanizm jest taki sam. Będziemy go określali jako *adresowanie z przesunięciem* (*displacement adressing*):

$$EA = A + (R)$$

Adresowanie z przesunięciem wymaga dwóch pól adresowych w rozkazie, przy czym przynajmniej jedno z nich jest jawne. Wartość zawarta w jednym polu adresowym (wartość równa A) jest używana bezpośrednio. Drugie pole adresowe, lub odniesienie domyślne oparte na kodzie operacji, odnosi się do rejestru, którego zawartość po dodaniu do A daje adres efektywny.

Opiszemy trzy najczęstsze zastosowania adresowania z przesunięciem:

- adresowanie względne,
- adresowanie z rejestrem podstawowym,
- indeksowanie.

Adresowanie względne

W przypadku adresowania względnego rejestrem domyślnym jest licznik programu (PC). Oznacza to, że adres bieżącego rozkazu jest dodawany do pola adresowego w celu otrzymania EA. Zwykle dla celów tej operacji pole adresowe jest traktowane jako liczba w notacji uzupełnienia do dwóch. Wobec tego adres efektywny jest przesunięty względem adresu zawartego w rozkazie.

Adresowanie względne opiera się na koncepcji lokalności przedyskutowanej w rozdz. 4 i 8. Jeśli większość odniesień do pamięci znajduje się stosunkowo blisko wykonywanego rozkazu, to używanie adresowania względnego pozwala na zaoszczędzenie bitów adresowych w rozkazie.

Adresowanie z rejestrem podstawowym

Interpretacja adresowania z rejestrem podstawowym jest następująca. Rejestr, do którego się odwołujemy, zawiera adres pamięci, a pole adresowe zawiera przesunięcie (zwykle w postaci liczby całkowitej bez znaku) w stosunku do tego adresu. Odniesienie do rejestru może być jawne lub domyślne.

Adresowanie z rejestrem podstawowym również wykorzystuje lokalność odniesień do pamięci. Jest wygodnym środkiem wdrażania segmentowania, które zostało przedyskutowane w rozdz. 8. W niektórych implementacjach jest stosowany pojedynczy rejestr podstawowy segmentu, co umożliwia stosowanie odniesień domyślnych. W innych implementacjach programista może wybrać rejestr przechowujący adres podstawowy segmentu, a rozkaz musi się do niego odnosić w sposób jawny. W tym ostatnim przypadku, jeśli długość pola adresowego wynosi K, a liczba możliwych rejestrów jest N, to jeden rozkaz może odnosić się do jednego z N obszarów zawierających po 2^K słów.

Indeksowanie

Interpretacja indeksowania jest zwykle następująca. Pole adresowe odnosi się do adresu w pamięci głównej, a wywołany rejestr zawiera dodatnie przesunięcie w stosunku do tego adresu. Zauważmy, że to rozwiązanie jest odwrotne w stosunku do adresowania z rejestrem podstawowym. Oczywiście chodzi o więcej niż tylko o interpretację użytkownika. Ponieważ w przypadku indeksowania pole adresowe jest traktowane jako adres w pamięci, zawiera ono zwykle więcej bitów niż pole adresowe w porównywalnym rozkazie dotyczącym rejestru podstawowego. Zobaczymy także, że istnieją pewne ulepszenia, które nie byłyby tak użyteczne w kontekście rejestru podstawowego. Mimo to metoda obliczania EA jest taka sama w przypadkach indeksowania i adresowania z rejestrem podstawowym. W obu odniesienie do rejestru jest czasem jawne, a czasem domyślne (w różnych typach procesorów).

Ważnym zastosowaniem indeksowania jest umożliwienie efektywnego prowadzenia operacji iteracyjnych. Rozważmy na przykład listę liczb przechowywanych począwszy od lokacji A. Załóżmy, że chcemy dodać 1 do każdego elementu listy. Musimy pobrać każdą wartość, dodać do niej 1 i zwrócić ją do pamięci. Sekwencja efektywnych adresów, jakiej potrzebujemy, jest następująca: A, A + 1, A + 2, ..., aż do ostatniej lokacji listy. Przy indeksowaniu można to łatwo uzyskać. Wartość A jest przechowywana w polu adresowym rozkazu, a wybrany rejestr, nazywany *rejestrem indeksowym*, jest inicjowany na 0. Po każdej operacji zawartość rejestru indeksu jest zwiększana o 1.

Ponieważ rejestry indeksowe są powszechnie używane do zadań iteracyjnych tego rodzaju, potrzeba zwiększania lub zmniejszania stanu rejestru indeksowego po każdym odniesieniu jest częsta. Ponieważ jest to tak powszechna operacja, niektóre systemy wykonują ją automatycznie, jako część tego samego cyklu rozkazu. Jest to znane jako *autoindeksowanie*. Jeśli pewne rejestry są przypisane wyłącznie do inde-

sowania, to autoindeksowanie może być wywoływane domyślnie i automatycznie. Jeśli są używane rejestry robocze, operacja autoindeksowania może wymagać sygnalizowania za pomocą bitu rozkazu. Autoindeksowanie oparte na przyrostach można przedstawić następująco:

$$EA = A + (R)$$

$$(R) \leftarrow (R) + 1$$

W niektórych maszynach jest możliwe zarówno adresowanie pośrednie, jak i indeksowanie. Możliwe jest też wykorzystywanie obu w tym samym rozkazie. Wówczas indeksowanie może być stosowane albo przed adresowaniem pośrednim, albo po nim.

Jeśli indeksowanie jest zastosowane po adresowaniu pośrednim, to jest określane jako *indeksowanie wtórne* (*postindexing*):

$$EA = (A) + (R)$$

Najpierw zawartość pola adresowego jest użyta do uzyskania dostępu do lokacji pamięci zawierającej adres bezpośredni. Adres ten jest następnie indeksowany za pomocą wartości rejestru. Metoda ta jest użyteczna, jeśli chcemy uzyskać dostęp do jednego z pewnej liczby bloków danych o ustalonym formacie. Na przykład, jak mówiliśmy w rozdz. 8, system operacyjny dla każdego procesu wykorzystuje blok sterowania procesem. Wykonywane operacje są takie same niezależnie od tego, który blok podlega obróbce. Wobec tego adresy w rozkazach odnoszących się do bloku powinny wskazywać lokację (wartość = A) zawierającą zmienny wskaźnik początku bloku sterowania procesem. Rejestr indeksowy zawiera przemieszczenie wewnątrz bloku.

W przypadku *indeksowania wstępnego* (*preindexing*) indeksowanie jest realizowane przed adresowaniem pośrednim:

$$EA = (A + (R))$$

Adres jest obliczany tak jak przy prostym indeksowaniu. Jednak w tym przypadku obliczany adres nie zawiera argumentu, lecz adres argumentu. Przykładem użycia tej metody jest tworzenie wielodrożnych tablic rozgałęzień. W określonym punkcie programu może następować rozgałęzienie do jednej z wielu lokacji, zależnie od warunków. Tablica adresów może być utworzona począwszy od lokacji A. Drogą indeksowania może być odnaleziona wymagana lokacja w tablicy.

Zwykle lista rozkazów nie obejmuje ani indeksowania wstępnego, ani wtórnego.

Adresowanie stosowe

Ostatnim trybem adresowania, jaki rozpatrzymy, jest adresowanie stosowe. Jak zdefiniowaliśmy w dodatku 10A, stos jest liniową tablicą lokacji. Jest czasem określany jako lista „spychania” (*pushdown list*) lub *kolejka ostatni na wejściu, pierwszy na*

wyjściu. Stos jest zarezerwowanym blokiem lokacji. Elementy są wprowadzane na wierzchołek stosu w ten sposób, że w określonym czasie blok jest częściowo wypełniony. Ze stosem jest związany wskaźnik, którego wartością jest adres wierzchołka stosu. Dwa wierzchołkowe elementy stosu mogą też znajdować się w rejestrach procesora, a wtedy wskaźnik stosu odnosi się do trzeciego elementu stosu (rys. 10.14b). Wskaźnik stosu jest przechowywany w rejestrze. Wobec tego odniesienia do lokacji w stosie są w rzeczywistości bezpośrednimi adresami rejestru.

Stosowy tryb adresowania jest formą adresowania domyślnego. Rozkazy maszynowe nie muszą zawierać odniesienia do pamięci, lecz domyślnie operują na wierzchołku stosu.

11.2. Tryby adresowania Pentium i PowerPC

Tryby adresowania Pentium

Przypomnijmy sobie na podstawie rys. 8.21, że mechanizm translacji adresu procesora Pentium tworzy adres zwany wirtualnym lub efektywnym, który jest adresem względnym w segmencie. Suma adresu początkowego segmentu i adresu efektywnego daje adres liniowy. Jeśli jest używane stronicowanie, adres liniowy musi przejść przez mechanizm translacji strony, w wyniku czego otrzymuje się adres fizyczny. W dalszym ciągu będziemy ignorowali ten ostatni krok, ponieważ jest on przezroczysty dla listy rozkazów i dla programisty.

Procesor Pentium jest wyposażony w różne tryby adresowania w celu umożliwienia efektywnej realizacji programów w językach wysokiego poziomu. Na rysunku 11.2 jest pokazany schemat adresowania. Segment będący przedmiotem odniesienia jest określony przez rejestr segmentowy. Istnieje sześć rejestrów segmentowych; użycie jednego z nich do określonego odniesienia zależy od rozkazu i od kontekstu jego wykonywania. Każdy rejestr segmentowy zawiera adres początkowy odpowiedniego segmentu. Z każdym widzialnym dla użytkownika rejestrem segmentowym jest związany rejestr deskryptora segmentu (niewidzialny dla programisty), który zawiera zapis praw dostępu do segmentu, a także adres początkowy i granicę (długość) segmentu. Ponadto istnieją dwa rejestry, które mogą być użyte do budowania adresu: rejestr podstawowy i rejestr indeksowy (*index register*).

W tabeli 11.2 jest wymienionych 12 trybów adresowania Pentium. Rozpatrzmy kolejno każdy z nich.

W **trybie natychmiastowym** argument jest włączony do rozkazu. Argument może być bajtem, słowem lub podwójnym słowem danych.

W **trybie argumentu rejestrowego** (trybie rejestrowym) argument jest umieszczony w rejestrze. Jeśli mamy do czynienia z rozkazami ogólnymi, takimi jak rozkazy przesyłania danych oraz rozkazy arytmetyczne i logiczne, argument może być umieszczany w jednym z 32-bitowych rejestrów roboczych (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP), w jednym z 16-bitowych rejestrów roboczych (AX, BX, CX, DX, SI,

DI, SP, BP) lub w jednym z 8-bitowych rejestrów roboczych (AH, BH, CH, DH, AL, BL, CL, DL). Do celów operacji zmiennopozycyjnych argumenty 64-bitowe są formowane w pary za pomocą dwóch rejestrów 32-bitowych. Są również pewne rozkazy, które odwołują się do rejestrów segmentowych (CS, DS, ES, SS, FS, GS).

Tabela 11.2. Tryby adresowania procesora Pentium

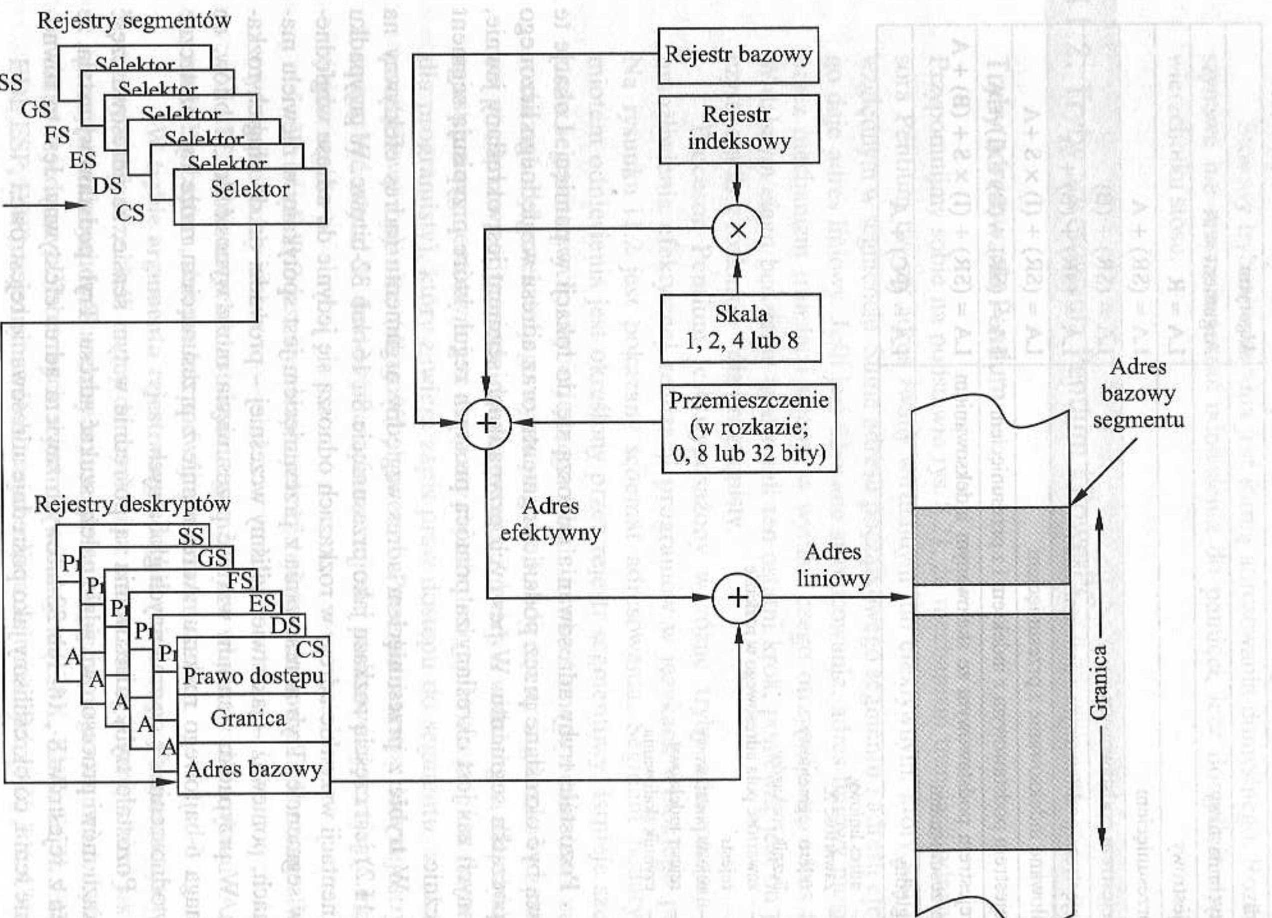
Tryb	Algorytm
Natychmiastowy	Argument = A
Rejestrowy	LA = R
Z przesunięciem	LA = (SR) + A
Z rejestrem podstawowym	LA = (SR) + (B)
Z rejestrem podstawowym z przesunięciem	LA = (SR) + (B) + A
Skalowane indeksowanie z przesunięciem	LA = (SR) + (I) × S + A
Z rejestrem podstawowym z indeksem i z przesunięciem	LA = (SR) + (B) + (I) + A
Z rejestrem podstawowym ze skalowanym indeksowaniem i z przesunięciem	LA = (SR) + (I) × S + (B) + A
Względny	LA = (PC) + A

LA – adres liniowy
(X) – zawartość X
SR – rejestr segmentowy
PC – licznik rozkazów
A – zawartość pola adresowego w rozkazie
R – rejestr
B – rejestr podstawowy
I – rejestr indeksowy
S – czynnik skalowania

Pozostałe tryby adresowania odnoszą się do lokacji w pamięci. Lokacje te muszą być określane przez podanie segmentu oraz adresu względnego liczonego od początku segmentu. W pewnych przypadkach segment jest określany jawnie, w innych zaś jest określany za pomocą prostych reguł, które przypisują segment zaocznie.

W **trybie z przesunięciem** adres względny argumentu (adres efektywny na rys. 11.2) jest częścią rozkazu jako przesunięcie 8-, 16- lub 32-bitowe. W przypadku segmentacji wszystkie adresy w rozkazach odnoszą się jedynie do adresu względnego w segmencie. Tryb adresowania z przesunięciem jest spotykany w niewielu maszynach, ponieważ – jak stwierdziliśmy wcześniej – prowadzi on do długich rozkazów. W przypadku Pentium wartość przesunięcia może wynosić do 32 bitów, co wymaga 6-bajtowego rozkazu. Adresowanie z przesunięciem może być użyteczne przy odnoszeniu się do zmiennych globalnych.

Pozostałe tryby adresowania są pośrednie w tym sensie, że adresowa część rozkazu mówi procesorowi, gdzie należy szukać adresu. **Tryb podstawowy** ustala, że jeden z rejestrów 8-, 16- lub 32-bitowych zawiera adres efektywny. Jest to równoważne temu, co określiliśmy jako pośrednie adresowanie rejestrowe.



Rysunek 11.2. Obliczanie trybu adresowania Pentium

W trybie **podstawowym z przesunięciem** rozkaz zawiera przesunięcie, które jest dodawane do rejestru podstawowego. Rejestrem podstawowym może być dowolny z rejestrów roboczych. Oto przykłady stosowania tego trybu:

- Jest używany przez kompilator do wyznaczania początku obszaru zmiennej lokalnej. Na przykład rejestr podstawowy może wskazać początek ramki stosu, która zawiera zmienne lokalne odpowiedniej procedury.
- Jest używany do indeksowania tablicy, której elementy mają rozmiar różny od 1, 2, 4 lub 8 bajtów i która wobec tego nie może być indeksowana za pomocą rejestru indeksowego. W tym przypadku przesunięcie wskazuje początek tablicy, a rejestr podstawowy zawiera wyniki obliczeń określające adres względny elementu wewnątrz tablicy.
- Jest używany w celu uzyskiwania dostępu do pola rekordu. Rejestr podstawowy wskazuje początek rekordu, a przesunięcie jest adresem względnym pola.

W trybie **skalowanego indeksowania z przesunięciem** rozkaz zawiera przesunięcie, które jest dodawane do rejestru, w tym przypadku nazywanego *rejestrem indeksowym*. Rejestrem indeksowym może być dowolny z rejestrów roboczych z wyjątkiem ESP, który jest na ogół używany do przetwarzania stosowego. Obliczanie adresu efektywnego polega na pomnożeniu zawartości rejestru indeksowego przez czynnik 1, 2, 4 lub 8, a następnie na dodaniu wyniku do przesunięcia. Tryb ten jest bardzo wygodny do indeksowania tablic. Czynnik skalowania 2 może być użyty w przypadku tablic 16-bitowych liczb całkowitych, czynnik 4 – w przypadku 32-bitowych liczb całkowitych lub zmiennopozycyjnych. Wreszcie czynnik 8 może być użyty w przypadku tablic liczb zmiennopozycyjnych o podwójnej dokładności.

W trybie **indeksowym podstawowym z przesunięciem** sumuje się zawartość rejestru podstawowego, rejestru indeksowego oraz przesunięcia w celu utworzenia adresu efektywnego. Znów rejestrem podstawowym może być dowolny z rejestrów roboczych, a rejestrem indeksowym – dowolny z rejestrów roboczych z wyjątkiem ESP. Tryb ten może być na przykład używany do adresowania lokalnej tablicy w ramce stosu. Może być także używany do obsługi tablic dwuwymiarowych; w tym przypadku przesunięcie wskazuje początek tablicy, a każdy rejestr dotyczy jednego wymiaru tablicy.

W trybie **podstawowym z przesunięciem i ze skalowanym indeksowaniem** sumuje się zawartość rejestru indeksowego, pomnożoną przez czynnik skalowania, z zawartością rejestru podstawowego i przesunięciem. Jest to użyteczne, gdy tablica jest przechowywana w ramce stosu; w tym przypadku długość elementów tablicy powinna wynosić po 2, 4 lub 8 bajtów. Tryb ten zapewnia również efektywne indeksowanie tablicy dwuwymiarowej, gdy elementy tablicy mają długość po 2, 4 lub 8 bajtów.

Wreszcie **adresowanie względne** może być używane w rozkazach przekazywania sterowania. Przesunięcie jest dodawane do wartości licznika programu, który wskazuje następny rozkaz. W tym przypadku przesunięcie jest traktowane jako bajt, słowo lub podwójne słowo ze znakiem, a jego wartość albo zwiększa, albo zmniejsza adres w liczniku programu.

Tryby adresowania PowerPC

Podobnie jak większość maszyn RISC oraz w przeciwieństwie do Pentium i większości maszyn CISC, procesor PowerPC używa stosunkowo prostego zestawu trybów adresowania. Jak widać w tabeli 11.3, tryby te są dogodnie poklasyfikowane ze względu na rodzaj rozkazu.

Tabela 11.3. Tryby adresowania procesora PowerPC

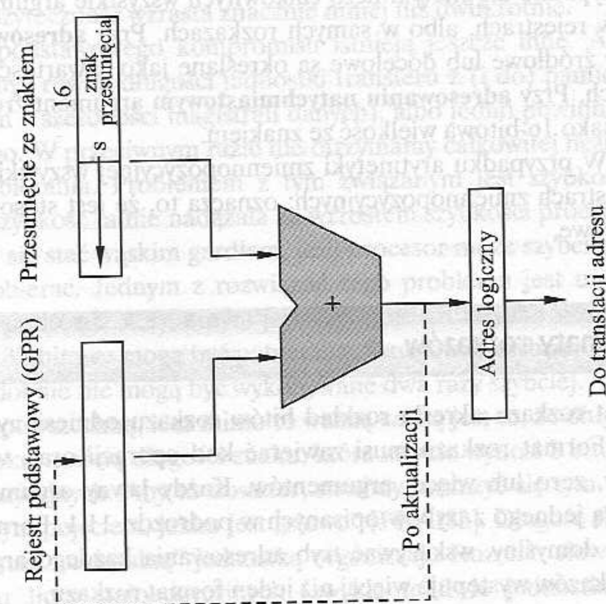
Tryb	Algorytm
Adresowanie ładowania/zapisu	
Pośredni	EA = (BR) + D
Pośredni indeksowany	EA = (BR) + (IR)
Adresowanie rozgałęzień	
Bezwzględny	EA = I
Względny	EA = (PC) + I
Pośredni	EA = (L/CR)
Obliczenia stałopozycyjne	
Rejestrowy	EA = GPR
Natychmiastowy	Argument = I
Obliczenia zmiennopozycyjne	
Rejestrowy	EA = FPR

EA – adres efektywny
(X) – zawartość X
BR – rejestr podstawowy
IR – rejestr indeksowy
L/CR – rejestr powiązań lub zliczający
GPR – rejestr roboczy
FPR – rejestr zmiennopozycyjny
D – przesunięcie
I – wartość natychmiastowa
PC – licznik rozkazów

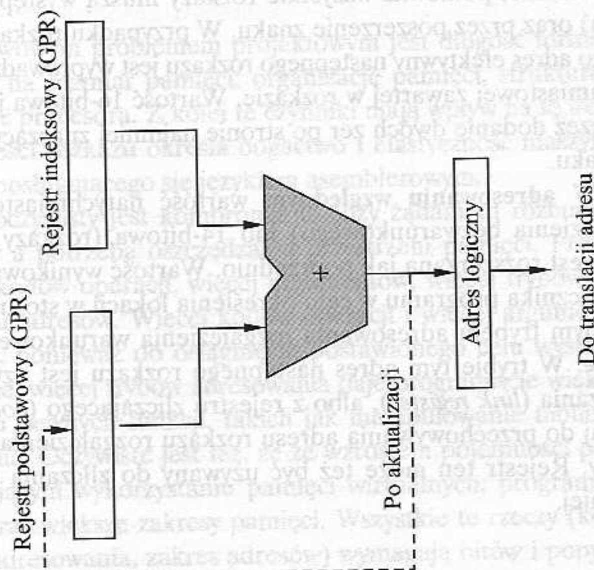
Adresowanie ładowania/zapisu

W procesorze PowerPC przewidziano dwa alternatywne tryby adresowania dla rozkazów ładowania/zapisu (rys. 11.3). W **adresowaniu pośrednim** rozkaz zawiera 16-bitowe przesunięcie dodawane do rejestru podstawowego, którym może być dowolny z rejestrów roboczych. Ponadto rozkaz może ustalać, że nowo obliczony adres efektywny ma być skierowany do rejestru podstawowego, aktualizując jego zawartość. Opcja aktualizacji jest użyteczna przy progresywnym indeksowaniu tablic w pętlach.

Inną metodą adresowania w rozkazach ładowania/zapisu jest **pośrednie adresowanie indeksowane**. W tym przypadku rozkaz odnosi się do rejestru podstawowego i rejestru indeksowego, z których oba mogą być dowolnym rejestrem roboczym. Adres efektywny jest sumą zawartości tych dwóch rejestrów. Jak poprzednio opcja aktualizacji umożliwia wprowadzenie nowego adresu efektywnego do rejestru podstawowego.



(a) Adresowanie pośrednie



(b) Adresowanie pośrednie indeksowe

Rysunek 11.3. Tryby adresowania argumentów w pamięci PowerPC

Adresowanie rozgałęzienia

Przewidziane są trzy tryby adresowania rozgałęzienia. Gdy używane jest **adresowanie bezwzględne** w połączeniu z rozkazem rozgałęzienia bezwarunkowego, adres efektywny następnego rozkazu jest otrzymywany z 24-bitowej wartości natychmiastowej (*immediate value*) zawartej w rozkazie. Wartość ta jest rozszerzana do 32 bitów przez dodanie dwóch zer po stronie najmniej znaczących bitów (co jest dopuszczalne, ponieważ wszystkie rozkazy muszą występować w 32-bitowych granicach) oraz przez poszerzenie znaku. W przypadku rozkazów rozgałęzienia warunkowego adres efektywny następnego rozkazu jest wyprowadzany z 16-bitowej wartości natychmiastowej zawartej w rozkazie. Wartość 16-bitowa jest rozszerzana do 32 bitów przez dodanie dwóch zer po stronie najmniej znaczących bitów oraz rozszerzenie znaku.

W **adresowaniu względnym** wartość natychmiastowa 24-bitowa (rozkazy rozgałęzienia bezwarunkowego) lub 14-bitowa (rozkazy rozgałęzienia warunkowego) jest rozszerzana jak poprzednio. Wartość wynikowa jest następnie dodawana do licznika programu w celu określenia lokacji w stosunku do bieżącego rozkazu. Innym trybem adresowania rozgałęzienia warunkowego jest **adresowanie pośrednie**. W trybie tym adres następnego rozkazu jest uzyskiwany albo z rejestru powiązania (*link register*), albo z rejestru zliczającego (*count register*). Zauważmy, że tutaj do przechowywania adresu rozkazu rozgałęzienia jest używany rejestr zliczający. Rejestr ten może też być używany do zliczania w pętli, co wyjaśniliśmy wcześniej.

Rozkazy arytmetyczne

W przypadku arytmetyki liczb całkowitych wszystkie argumenty muszą być zawarte albo w rejestrach, albo w samych rozkazach. Przy **adresowaniu rejestrowym** argumenty źródłowe lub docelowe są określane jako zawartość jednego z rejestrów roboczych. Przy **adresowaniu natychmiastowym** argument źródłowy występuje w rozkazie jako 16-bitowa wielkość ze znakiem.

W przypadku arytmetyki zmiennopozycyjnej wszystkie argumenty są zawarte w rejestrach zmiennopozycyjnych; oznacza to, że jest stosowane tylko adresowanie rejestrowe.

11.3. Formaty rozkazów

Format rozkazu określa rozkład bitów rozkazu odniesiony do jego części składowych. Format rozkazu musi zawierać kod operacji oraz, w sposób jawny lub domyślny, zero lub więcej argumentów. Każdy jawny argument jest adresowany za pomocą jednego z trybów opisanych w podrozdz. 11.1. Format musi, w sposób jawny lub domyślny, wskazywać tryb adresowania każdego argumentu. W większości list rozkazów występuje więcej niż jeden format rozkazu.

Projektowanie formatu rozkazu jest sztuką złożoną i wdrożono dotychczas zdumiewającą różnorodność rozwiązań. W tym podrozdziale przeanalizujemy podstawowe zagadnienia projektowania, odwołując się krótko do pewnych przykładów, po czym szczegółowo przedstawimy rozwiązania przyjęte w procesorach Pentium i PowerPC.

Długość rozkazu

Najbardziej podstawowym problemem projektowym jest długość formatu rozkazu. Decyzja ta wpływa na rozmiar pamięci, organizację pamięci, strukturę magistrali, złożoność i szybkość procesora. Z kolei te czynniki mają wpływ na tę decyzję. Decyzja dotycząca długości rozkazu określa bogactwo i elastyczność maszyny, widziane przez programistę posługującego się językiem asemblerowym.

Najbardziej oczywisty jest kompromis między żądaniem rozbudowanego repertuaru rozkazów a potrzebą oszczędzania przestrzeni pamięci. Programiści domagają się więcej kodów operacji, więcej argumentów, więcej trybów adresowania i większego zakresu adresów. Więcej kodów operacji i więcej argumentów ułatwia życie programiście, ponieważ do osiągnięcia postawionego celu wystarczą krótsze programy. Podobnie, więcej trybów adresowania daje programiście większą elastyczność we wdrażaniu pewnych funkcji, takich jak manipulowanie tablicami i wielodrożne rozgałęzienia. Oczywiste jest też, że ze wzrostem pojemności pamięci głównej i przy wzrastającym wykorzystaniu pamięci wirtualnych, programista chciałby móc adresować coraz większe zakresy pamięci. Wszystkie te rzeczy (kody operacji, argumenty, tryby adresowania, zakres adresów) wymagają bitów i popychają w kierunku dłuższych rozkazów. Jednak dłuższe rozkazy mogą prowadzić do rozrzutności. Rozkaz 64-bitowy zajmuje dwa razy tyle przestrzeni co 32-bitowy, jednak prawdopodobnie jego użyteczność wzrasta znacznie mniej niż dwukrotnie.

Obok tego podstawowego kompromisu istnieją jeszcze inne. Albo długość rozkazu powinna być równa długości jednostki transferu z (i do) pamięci (w systemie magistralowym – szerokości magistrali danych), albo jedno powinno być wielokrotnością drugiego. W przeciwnym razie nie otrzymamy całkowitej liczby rozkazów podczas cyklu pobierania. Problemem z tym związanym jest szybkość transferu z (i do) pamięci. Szybkość ta nie nadążała za wzrostem szybkości procesora. Wobec tego pamięć może się stać wąskim gardłem, jeśli procesor może szybciej wykonywać rozkazy, niż je pobierać. Jednym z rozwiązań tego problemu jest użycie pamięci podręcznej (patrz podrozdz. 4.3); innym jest używanie krótszych rozkazów. Jak poprzednio, rozkazy 32-bitowe mogą być pobierane dwukrotnie szybciej niż 64-bitowe, jednak prawdopodobnie nie mogą być wykonywane dwa razy szybciej.

Ewidentnie powszechną, lecz mimo to ważną cechą jest to, że długość rozkazu powinna być wielokrotnością długości znaku, która zwykle wynosi 8 bitów, oraz długości liczb stałopozycyjnych. Aby to dostrzec, musimy posłużyć się tym tak nieszczęśliwie zdefiniowanym pojęciem, jakim jest „słowo" [FRAI83]. Długość słowa pamięci jest w pewnym sensie „naturalną" jednostką organizacji. Rozmiar słowa zwykle determinuje rozmiar liczb stałopozycyjnych (zwykle te dwie jednostki są równe).

Rozmiar słowa jest zwykle równy rozmiarowi jednostki transferu z/do pamięci albo przynajmniej jest z nim integralnie związany. Ponieważ powszechną formą danych są dane znakowe, chcielibyśmy dysponować słowem, w którym może być przechowywana całkowita liczba znaków. W przeciwnym razie przechowywanie wielu znaków pociągnie za sobą stratę bitów w każdym słowie albo znak będzie wykraczał poza granice słowa. Znaczenie tego problemu jest tak duże, że gdy IBM wprowadzał System/360 i chciał używać znaków 8-bitowych, podjął bolesną decyzję odejścia od 36-bitowej architektury naukowych komputerów z serii 700/7000 i przyjęcia architektury 32-bitowej.

Przydział bitów

Rozpatrzyliśmy niektóre z czynników wpływających na decyzje dotyczące długości formatu rozkazu. Równie trudnym problemem jest przydział bitów w tym formacie. Mamy tu do czynienia ze skomplikowanymi kompromisami.

Przy danej długości rozkazu istnieje oczywista wymienność między liczbą kodów operacji a zdolnością adresowania. Więcej kodów operacji oznacza oczywiście więcej bitów w polu kodu operacji. W przypadku formatu rozkazu o danej długości następuje zredukowanie liczby bitów dostępnych na potrzeby adresowania. Istnieje interesujące ulepszenie dotyczące tego problemu: jest nim użycie kodów operacji o zmiennej długości. W tym rozwiązaniu występuje minimalna długość kodu operacji, jednak dla niektórych kodów mogą być specyfikowane dodatkowe operacje za pomocą dodatkowych bitów w rozkazie. Przy rozkazach o ustalonej długości pozostawia to mniej bitów na adresowanie. Cecha ta jest wobec tego używana w odniesieniu do tych rozkazów, które wymagają mniej argumentów i (lub) mniej rozbudowanego adresowania.

Następujące wzajemnie powiązane czynniki muszą być uwzględniane przy decydowaniu o użyciu bitów na potrzeby adresowania:

- **Liczba trybów adresowania.** Czasem tryb adresowania może być domyślny. Na przykład pewne kody operacji mogą zawsze wymagać indeksowania. W innych przypadkach tryby adresowania muszą być jawne, do czego jest potrzebny jeden lub więcej bitów.
- **Liczba argumentów.** Widzieliśmy, że zmniejszona liczba adresów może prowadzić do dłuższych, bardziej kłopotliwych programów (np. rys. 10.3). Typowe rozkazy współczesnych maszyn przewidują dwa argumenty. Każdy adres argumentu w rozkazie może wymagać własnego wskaźnika trybu lub też wskaźnik trybu może być związany tylko z jednym polem adresowym.
- **Rejestry a pamięć.** Maszyna musi mieć rejestry, dzięki którym dane mogą być wprowadzane do procesora w celu przetwarzania. W przypadku rejestru pojedynczego, widzialnego dla użytkownika (nazywanego zwykle *akumulatorem*), adres jednego argumentu jest domyślny i nie pochłania bitów rozkazu. Jednak programowanie jednorejestrowe jest kłopotliwe i wymaga wielu rozkazów. Nawet w przypadku wielu rejestrów potrzeba tylko kilku bitów do określenia rejestru. Im więcej rejestrów można przeznaczyć na odniesienia do argumentów, tym

mniej potrzeba bitów. W wielu badaniach wykazano, że pożądana liczba rejestrów widzialnych dla użytkownika wynosi od 8 do 32 [LUND77], [HUCK83]. W większości współczesnych architektur występują przynajmniej 32 rejestry.

- **Liczba zestawów rejestrów.** Wiele maszyn ma jeden zestaw rejestrów roboczych, zawierający zwykle 32 lub więcej rejestrów. Mogą one być używane do przechowywania danych albo też do przechowywania adresów w przypadku adresowania z przesunięciem. W niektórych architekturach, włącznie z Pentium, występuje zbiór dwóch lub większej liczby wyspecjalizowanych zestawów (takich, jak przeznaczone do danych lub przesunięć). Zaletą tego rozwiązania jest to, że przy ustalonej liczbie rejestrów ich podział funkcjonalny oznacza mniej bitów potrzebnych w rozkazie. Na przykład przy dwóch zestawach po 8 rejestrów do identyfikowania rejestru potrzebne są tylko 3 bity; kod operacyjny domyślnie określa, do którego zestawu rejestrów następuje odniesienie.
- **Zakres adresów.** W przypadku adresów odnoszących się do pamięci zakres możliwych adresów jest powiązany z liczbą bitów adresowych. Ponieważ stanowi to poważne ograniczenie, bezpośrednie adresowanie jest rzadko stosowane. Dzięki adresowaniu z przesunięciem zakres adresów ulega poszerzeniu aż do wielkości wynikającej z długości rejestru adresowego. Nawet w tym przypadku nadal wygodne jest stosowanie raczej dużych przesunięć w stosunku do adresu rejestrowego, co wymaga stosunkowo dużej liczby bitów adresowych w rozkazie.
- **Stopień granulacji adresu.** W przypadku adresów, które odnoszą się raczej do pamięci niż do rejestrów, dodatkowym czynnikiem jest stopień granulacji (*granularity*) adresu. W systemie o słowach 16- lub 32-bitowych adres może odnosić się do słowa lub bajta, zależnie od wyboru projektanta. Adresowanie bajtowe jest wygodne do manipulowania znakami, jednak przy ustalonym rozmiarze pamięci wymaga więcej bitów adresowych.

Wobec tego projektant ma do czynienia z całym zastępem czynników, które musi przeanalizować i zrównoważyć. Nie jest jasne, jak krytyczne są różne wybory. Zacytujemy na przykład jedno z badań [CRAG79], w którym porównano różne rozwiązania formatu rozkazu, włącznie z użyciem stosu, rejestrów roboczych, akumulatora i rozkazów typu pamięć-rejestr. Stosując spójny zespół założeń, nie stwierdzono znaczących różnic przestrzeni kodowej lub czasu realizacji.

Zapoznajmy się krótko z równoważeniem tych różnych czynników, które zastosowano w dwóch projektach maszyn.

PDP-8

Jednym z najprostszych projektów rozkazów dla komputera uniwersalnego było rozwiązanie przyjęte dla PDP-8 [BELL78b]. Komputer PDP-8 używa rozkazów 12-bitowych i operuje na słowach 12-bitowych. Występuje w nim jeden rejestr roboczy – akumulator.

Mimo ograniczeń tego projektu, adresowanie jest całkiem elastyczne. Każde odniesienie do pamięci obejmuje 7 bitów plus 2 modyfikatory 1-bitowe. Pamięć jest

podzielona na strony o ustalonej długości, po $2^7 = 128$ słów każda. Obliczanie adresu opiera się na odniesieniu do strony 0 lub strony bieżącej (zawierającej dany rozkaz), zależnie od bitu strony. Drugi modyfikator 1-bitowy wskazuje, czy stosowane jest adresowanie bezpośrednie, czy pośrednie. Oba tryby mogą być używane w połączeniu, tak że adres pośredni jest 12-bitowym adresem zawartym w słowie strony 0 lub strony bieżącej. Ponadto, 8 dedykowanych słów na stronie 0 stanowi „rejestry" automatycznego indeksowania. Gdy ma miejsce pośrednie odniesienie do tych lokacji, następuje wstępne indeksowanie.

Na rysunku 11.4 jest pokazany format rozkazu PDP-8. Występuje tu 3-bitowy kod operacji i trzy rodzaje rozkazów. Przy kodach od 0 do 5 formatem jest rozkaz z 1-adresowym odniesieniem do pamięci, zawierający bit strony i bit pośredniości. Wobec tego występuje tylko 6 operacji podstawowych. W celu poszerzenia grupy

Rozkazy z odniesieniem do pamięci

Kod operacji	D/I	Z/C	Przemieszczenie
0 2	3	4	5 11

Rozkazy wejścia/wyjścia

1 1 0	Urządzenie	Kod operacji
0 2	3 8	9 11

Rozkazy z odniesieniem do rejestrów

Mikrorozkazy grupy 1

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Mikrorozkazy grupy 2

1	1	1	1	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Mikrorozkazy grupy 3

1	1	1	1	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

D/I – bezpośredni/pośredni adres
Z/C – strona zero lub strona bieżąca
CLA – wyczyść akumulator
CLL – wyczyść łącze
CMA – dopełnij akumulator
CML – dopełnij łącze
RAR – wykonaj rotację akumulatora w prawo
RAL – wykonaj rotację akumulatora w lewo
BSW – wymień bajt
IAC – inkrementuj akumulator
SMA – przeskocz na minus akumulator
SZA – przeskocz na zero akumulator
SNL – przeskocz na łącze niezerowe
RSS – odwróć odczyt przeskoku
OSR – wykonaj OR z rejestrem przełączania
HLT – zatrzymaj
MQA – mnożnik/iloraz do akumulatora
MQL – ładuj mnożnik/iloraz

Rysunek 11.4. Formaty rozkazu PDP-8

operacji kod operacji 7 określa odniesienie do rejestru lub *mikrorozkaz*. W tym formacie pozostałe bity służą do kodowania dodatkowych operacji. Na ogół każdy bit określa specyficzną operację (np. wyczyszczenie akumulatora), a bity te mogą być łączone w jednym rozkazie. Strategia mikroprogramowania była stosowana przez DEC jeszcze w PDP-1 i jest w pewnym sensie zapowiedzią dzisiejszych maszyn mikroprogramowanych, które będą przedyskutowane w części IV. Kod operacji 6 dotyczy wejścia-wyjścia; 6 bitów używa się do wyboru jednego z 64 urządzeń, a 3 bity określają konkretny rozkaz wejścia-wyjścia.

Efektywność formatu rozkazu PDP-8 jest godna uwagi. Umożliwia on adresowanie pośrednie, adresowanie z przesunięciem i indeksowanie. Wraz z rozszerzeniem kodu operacji obsługuje on łącznie około 35 rozkazów. Biorąc pod uwagę ograniczenia wynikające z 12-bitowego rozkazu, projektanci nie mogli spisać się lepiej.

PDP-10

Z listą rozkazów PDP-8 silnie kontrastuje lista PDP-10. PDP-10 został zaprojektowany jako duży system pracujący z podziałem czasu, z akcentem na łatwość programowania, nawet kosztem dodatkowych wydatków sprzętowych.

Do zasad wykorzystanych do zaprojektowania listy rozkazów należały [BELL78c]:

- **Ortogonalność.** Ortogonalność jest zasadą, zgodnie z którą dwie zmienne są od siebie niezależne. W kontekście listy rozkazów termin ten oznacza, że inne elementy rozkazu są niezależne od kodu operacji (nie są przez ten kod determinowane). Projektanci PDP-10 użyli tego terminu w celu podkreślenia faktu, że adres jest zawsze obliczany w ten sam sposób, niezależnie od kodu operacji. Stanowi to kontrast w stosunku do wielu maszyn, w których tryb adresowania wynika czasem domyślnie z użytego operatora.
- **Kompletność.** Każdy rodzaj danych arytmetycznych (całkowite, stałopozycyjne, rzeczywiste) powinien mieć kompletną i identyczną listę operacji.
- **Adresowanie bezpośrednie.** Zrezygnowano z adresowania przesunięciowego z rejestrem podstawowym, obciążającego programistę problemami organizacji pamięci, na rzecz adresowania bezpośredniego.

Głównym celem każdej z tych zasad było ułatwienie programowania.

PDP-10 ma słowo 36-bitowe oraz 36-bitowy rozkaz. Ustalony format rozkazu widać na rys. 11.5. Kod operacji zajmuje 9 bitów, co pozwala na 512 operacji. W rzeczywistości zdefiniowano 365 różnych rozkazów. Większość rozkazów ma dwa adresy, z których jeden znajduje się w jednym z 16-bitowych rejestrów roboczych. Wobec

Kod operacji	Rejestr	I	Rejestr indeksowy	Adres w pamięci
0 – 8	9 – 12	13	14 – 17	18 – 35

I = bit pośredności

Rysunek 11.5. Format rozkazu PDP-10

tego odniesienie do tego argumentu zajmuje 4 bity. Odniesienie do drugiego argumentu rozpoczyna się od 18-bitowego pola adresu pamięci. Może być ono używane jako argument natychmiastowy lub jako adres pamięci. W tym ostatnim przypadku dozwolone jest indeksowanie oraz adresowanie pośrednie. Te same rejestry robocze są używane jako rejestry indeksowe.

Rozkaz długości 36-bitów jest rzeczywistym luksusem. Nie ma potrzeby sprytnych rozwiązań zwiększających liczbę kodów operacji; 9-bitowe pole kodu jest więcej niż wystarczające. Adresowanie jest także proste. Atrakcyjne jest adresowanie bezpośrednie ze względu na 18-bitowe pole adresowe. Dla pamięci większych niż 2^{18} przewidziano adresowanie pośrednie. Aby ułatwić programowanie, istnieje możliwość indeksowania przy manipulowaniu tablicami i wykonywaniu programów iteracyjnych. Ponieważ pole argumentu ma 18 bitów, staje się atrakcyjne adresowanie natychmiastowe.

Projekt listy rozkazów PDP-10 osiągnął zapowiedziane cele [LUND77]. Lista jest stosunkowo łatwa z punktu widzenia programisty, kosztem nieefektywnego wykorzystania przestrzeni. Był to świadomy wybór dokonany przez projektantów i dlatego nie może być dyskwalifikowany jako słaby projekt.

Rozkazy o zmiennej długości

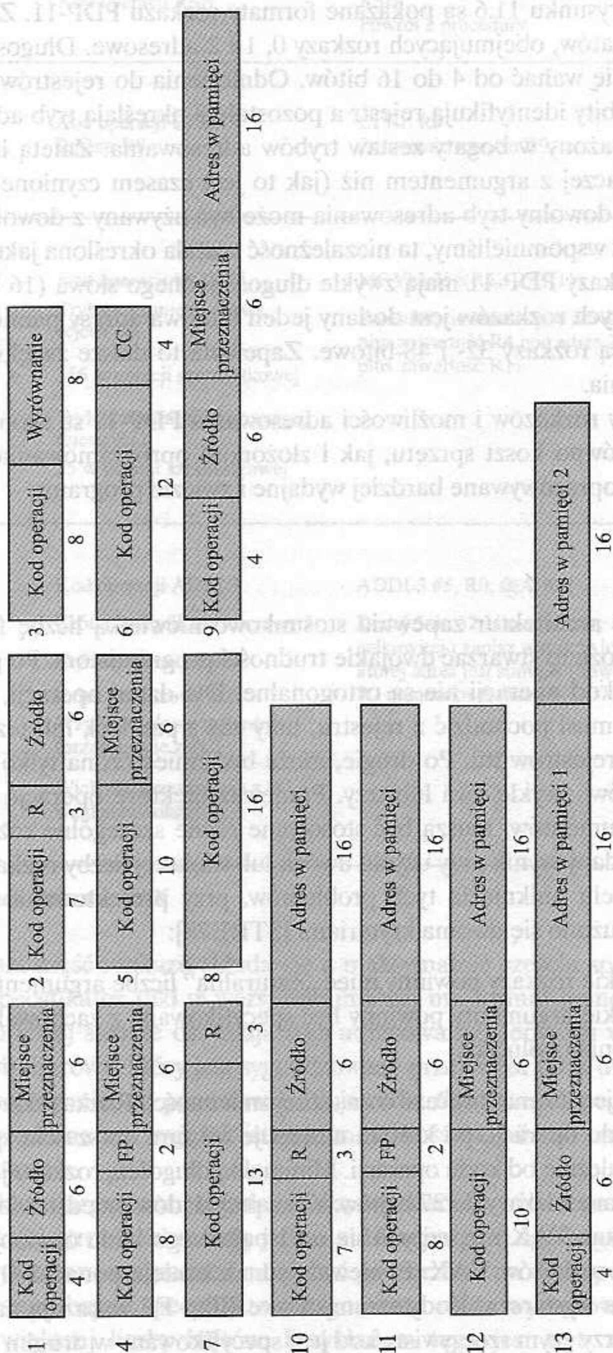
W dotychczasowych przykładach występowała jedna, ustalona długość rozkazów; przedyskutowane kompromisy dotyczyły tego właśnie kontekstu. Jednak projektanci mogą wybrać różne formaty rozkazów o różnych długościach. Taka taktyka ułatwia uzyskanie dużego repertuaru kodów operacji, o różnych długościach kodów. Adresowanie może być bardziej elastyczne przy różnych kombinacjach odniesień do rejestrów i do pamięci oraz przy różnych trybach adresowania. Rozkazy o zmiennej długości ułatwiają efektywne i zwarte wprowadzanie tych odmian.

Ceną, jaką płacimy za przyjęcie rozkazów o zmiennej długości, jest wzrost złożoności procesora. Malejące ceny sprzętu, zastosowanie mikroprogramowania (przedyskutowane w części IV) oraz ogólnie lepsze rozumienie zasad projektowania procesorów przyczyniły się do tego, że tę niewielką cenę warto zapłacić. Przekonamy się jednak, że komputery RISC i superskalarne mogą się posługiwać rozkazami o stałej długości w celu zwiększenia wydajności.

Zastosowanie rozkazów o zmiennej długości nie eliminuje żądania, aby wszystkie długości rozkazów były integralnie związane z długością słowa. Ponieważ procesor nie zna długości następnego rozkazu przewidzianego do pobrania, typową strategią jest pobieranie liczby bajtów lub słów równej przynajmniej najdłuższemu możliwemu rozkazowi. Oznacza to, że czasem pobiera się wiele rozkazów. Jak jednak zobaczymy w rozdz. 12, jest to w dowolnym przypadku strategia właściwa.

PDP-11

PDP-11 został zaprojektowany w celu zapewnienia skutecznej i elastycznej listy rozkazów w ramach ograniczeń minikomputera 16-bitowego [BELL70].



Rysunek 11.6. Formaty rozkazu stosowane w PDP-11

PDP-11 wykorzystuje zestaw ośmiu 16-bitowych rejestrów roboczych. Dwa z nich mają dodatkowe znaczenie: jeden jest używany jako wskaźnik stosu dla specjalnych operacji na stosie, a drugi pełni rolę licznika programu, który zawiera adres następnego rozkazu.

Na rysunku 11.6 są pokazane formaty rozkazu PDP-11. Zastosowano 13 różnych formatów, obejmujących rozkazy 0, 1 i 2-adresowe. Długość kodu operacyjnego może się wahać od 4 do 16 bitów. Odniesienia do rejestrów mają długość 6 bitów. Trzy bity identyfikują rejestr a pozostałe 3 określają tryb adresowania. PDP-11 jest wyposażony w bogaty zestaw trybów adresowania. Zaletą łączenia trybu adresowania raczej z argumentem niż (jak to jest czasem czynione) z kodem operacji jest to, że dowolny tryb adresowania może być używany z dowolnym kodem operacji. Jak już wspomnieliśmy, ta niezależność została określona jako *ortogonalność*.

Rozkazy PDP-11 mają zwykle długość jednego słowa (16 bitów). W przypadku niektórych rozkazów jest dodany jeden lub dwa adresy pamięci, więc częścią repertuaru są rozkazy 32- i 48-bitowe. Zapewnia to dalsze zwiększenie elastyczności adresowania.

Listy rozkazów i możliwości adresowania PDP-11 są skomplikowane. Zwiększa to zarówno koszt sprzętu, jak i złożoność oprogramowania. Zaletą jest to, że mogą być opracowywane bardziej wydajne i zwięzłe programy.

VAX

Większość architektur zapewnia stosunkowo niewielką liczbę formatów instrukcji stałych. Może to stwarzać dwojakie trudności programistom. Po pierwsze, tryb adresowania i kod operacji nie są ortogonalne. Dla danej operacji, na przykład, jeden argument musi pochodzić z rejestru, inny zaś z pamięci, lub też obydwa mogą pochodzić z rejestrów itd. Po drugie, może być zmieszczona tylko ograniczona liczba argumentów: zwykle dwa lub trzy. Ponieważ niektóre operacje z natury wymagają więcej argumentów, muszą być stosowane różne szczególne rozwiązania, aby osiągnąć pożądany wynik przy użyciu dwóch lub większej liczby rozkazów.

W celu uniknięcia tych problemów, przy projektowaniu formatu rozkazów VAX posłużono się dwoma kryteriami [STRE78]:

1. Wszystkie rozkazy powinny mieć „naturalną" liczbę argumentów.
2. Wszystkie argumenty powinny być specyfikowane z zachowaniem takiego samego stopnia ogólności.

Wynikiem jest format rozkazu o znacznej zmienności. Rozkaz składa się z 1- lub 2-bajtowego kodu operacji, po którym następuje od zera do sześciu specyfikatorów argumentów, zależnie od kodu operacji. Minimalną długością rozkazu jest jeden bajt; mogą być budowane rozkazy do 37 bajtów. Kilka przykładów przedstawiono na rys. 11.7.

Rozkaz VAX rozpoczyna się od 1-bajtowego kodu operacji. Wystarcza to dla większości rozkazów VAX. Ponieważ jednak istnieje ponad 300 różnych rozkazów, 8 bitów nie wystarcza. Kody szesnastkowe FD i FF wskazują na poszerzenie kodu operacji, przy czym rzeczywisty kod jest specyfikowany w drugim bajcie.

Format szesnastkowy	Objaśnienie	Notacja asemblerowa i opis
8 bitów 0 5	Kod operacji RSB	RSB Powrót z procedury
D 4	Kod operacji CLRL	CLRL R9
5 9	Rejestr R9	Zerowanie rejestru R9
B 0	Kod operacji MOVW	MOVM 356(R4), 25(R11)
C 4	Tryb przesunięcia słowa, rejestr R4	Przesunięcie słowa spod adresu 356 plus zawartość R4 pod adres 25 plus zawartość R11
6 4 0 1	356 w notacji szesnastkowej	
A B	Tryb przesunięcia bajtowego, rejestr R11	
1 9	25 w notacji szesnastkowej	
C 1	Kod operacji ADDL3	ADDL3 #5, R0, @A[R2]
0 5	Krótki specyfikator literowy 5	Dodaj 5 do 32-bitowej liczby całkowitej i zapisz wynik w lokacji, której adres jest sumą A i zawartości R2 pomnożonej przez 4
5 0	Tryb rejestrowy R0	
4 2	Index prefiksowy R2	
D F	Względne słowo pośrednie (przesunięcie z PC)	
	Wielkość przesunięcia z PC względem lokacji A	

Rysunek 11.7. Przykłady rozkazów VAX

Pozostała część rozkazu składa się z maksymalnie sześciu specyfikatorów argumentów. Specyfikator taki w wersji minimalnej ma format jednobajtowy, w którym 4 bity po lewej stronie określają tryb adresowania. Jedynym wyjątkiem od tej reguły jest tryb literowy, który jest sygnalizowany przez wzór 00 w dwóch pierwszych bitach po lewej stronie, co pozostawia miejsce dla 6-bitowego specyfikatora literowego. Ze względu na ten wyjątek istnieje możliwość określania łącznie 12 różnych trybów adresowania.

Specyfikator argumentu składa się często zaledwie z jednego bajta, przy czym 4 bity po prawej stronie określają jeden z 16 rejestrów ogólnego przeznaczenia. Długość specyfikatora argumentu może być poszerzona na dwa sposoby. Po pierwsze, po pierwszym bajcie specyfikatora argumentu może występować wartość stała jednego lub większej liczby bajtów. Przykładem tego jest tryb z przesunięciem,

w którym jest używane przesunięcie 8-, 16- lub 32-bitowe. Po drugie, może być użyty indeksowy tryb adresowania. W tym przypadku pierwszy bajt specyfikatora argumentu składa się z 4-bitowego kodu trybu adresowania 0100 oraz z 4-bitowego identyfikatora rejestru indeksowego. Pozostała część specyfikatora argumentu składa się ze specyfikatora adresu podstawowego, który może mieć długość jednego lub wielu bajtów.

Czytelnik – podobnie jak autor – może się zastanawiać, jaki rodzaj rozkazów może wymagać sześciu argumentów. Co zaskakujące, VAX ma cały szereg takich rozkazów. Rozważmy

ADDP6 OP1, OP2, OP3, OP4, OP5, OP6

Rozkaz ten dodaje dwie upakowane liczby dziesiętne. OP1 i OP2 określają długość i adres początkowy jednego łańcucha dziesiętnego; OP3 i OP4 określają drugi łańcuch. Obydwa te łańcuchy są dodawane, wynik zaś jest zapisywany w postaci łańcucha dziesiętnego, którego długość i lokacja początkowa są określane przez OP5 i OP6.

Lista rozkazów VAX zapewnia szeroką różnorodność operacji i trybów adresowania. Daje to programiście – na przykład piszącemu kompilatory – bardzo potężne i elastyczne narzędzie opracowywania programów. W teorii powinno to prowadzić do skutecznego kompilowania na język maszynowy programów napisanych w języku wysokiego poziomu oraz, ogólnie rzecz biorąc, do skutecznego i efektywnego wykorzystania zasobów procesora. Jednak ceną, jaka musi być zapłacona za te korzyści, jest znaczna złożoność procesora w porównaniu z tymi, które posługują się prostszym zbiorem i formatem rozkazów.

Wrócimy do tych zagadnień w rozdziale 13, w którym przeanalizujemy przypadek bardzo prostych zbiorów rozkazów.

11.4. Formaty rozkazów Pentium i PowerPC

Formaty rozkazów Pentium

Pentium został wyposażony w różnorodne formaty rozkazów. Spośród elementów opisanych poniżej, zawsze obecne jest tylko pole kodu operacji. Na rysunku 11.8 jest pokazany ogólny format rozkazu. Rozkazy są złożone z od 0 do 4 opcjonalnych przedrostków, z 1 lub 2-bajtowego kodu operacji, z opcjonalnego specyfikatora adresu składającego się z bajta ModR/m i bajta Scale Index, z opcjonalnego przesunięcia oraz z opcjonalnego pola natychmiastowego.

Rozpatrzmy najpierw bajty przedrostka:

- **Przedrostki rozkazu.** Przedrostek rozkazu, jeśli jest obecny, zawiera przedrostek LOCK lub jeden z *przedrostków powtarzania* (*repeat prefixes*). Przedrostek LOCK jest używany w celu zapewnienia wyłącznego użycia pamięci wspólnej w środowisku wieloprocesorowym. Przedrostki powtarzania określają powtarzalną operację na łańcuchu, która pozwala Pentium przetwarzać łańcuchy znacznie szybciej

Rysunek 11.8. Format rozkazu procesora Pentium

niż za pomocą zwykłych pętli programowych. Istnieje pięć różnych przedrostków powtarzania: REP, REPE, REPZ, REPNE i REPNZ. Gdy jest obecny bezwzględny przedrostek REP, operacja określona w rozkazie jest prowadzona powtarzalnie na kolejnych elementach łańcucha; liczba powtórzeń jest określona w rejestrze CX. Przedrostek warunkowy REP powoduje powtarzanie operacji, aż stan CX wyniesie zero lub do czasu spełnienia określonego warunku.

- **Pominięcie segmentu.** Jawne ustalenie, którego rejestru segmentu powinien użyć rozkaz, niezależnie od wyboru dokonanego przez Pentium w odniesieniu do tego rozkazu.
- **Rozmiar adresu.** Procesor może adresować pamięć, stosując adresy 16- lub 32-bitowe. Rozmiar adresu określa rozmiar przemieszczenia w rozkazach i rozmiar adresu względnego generowanego przy obliczaniu adresu efektywnego. Jeden z tych rozmiarów jest oznaczany jako pomijany, a przedrostek rozmiaru adresu powoduje przeskok między generowaniem adresów 16- a 32-bitowych.
- **Rozmiar argumentu.** Podobnie jak wyżej, w rozkazie pomijana jest długość argumentu 16 lub 32 bity, a przedrostek argumentu powoduje przeskok między argumentami 32- a 16-bitowymi.

Sam rozkaz zawiera następujące pola:

- **Kod operacji** (*opcode*). Jest on 1 lub 2-bajtowy. Kod operacji może także zawierać bity, które precyzują, czy dane są bajtowe, czy pełnowymiarowe (16 lub 32 bity zależnie od kontekstu), bity określające kierunek przesyłania danych (do lub z pamięci) oraz bity wskazujące, czy pole danych natychmiastowych musi być poszerzone o znak.
- **ModR/m.** Ten bajt oraz następny dostarczają informacji adresowej. Bajt ModR/m określa, czy argument znajduje się w rejestrze, czy w pamięci. Jeśli znajduje się w pamięci, to pola wewnątrz bajta identyfikują przewidziany do użytku tryb adresowania. Bajt ModR/m składa się z trzech pól: pole Mod (2 bity) łączy się z polem R/M, tworząc 32 możliwe wartości: 8 rejestrów i 24 tryby adresowania; pole „Rejestr/kod operacji" (3 bity) określa albo numer rejestru, albo daje 3 dodatkowe bity kodu operacji; pole „R/M" (3 bity) może wskazywać rejestr jako lokalizację argumentu lub może stanowić część kodu trybu adresowania w połączeniu z polem Mod.
- **Skalowanie-indeks-podstawa** (*SIB*). Jeden z kodów bajta ModR/m oznacza włączenie bajta SIB w celu pełnego określenia trybu adresowania. Bajt SIB składa się z trzech pól: pole SS (2 bity) identyfikuje czynnik skalowania przy skalowanym indeksowaniu, pole „indeks" (3 bity) – rejestr indeksowy, a pole Baza (3 bity) – rejestr podstawowy.
- **Przesunięcie** (*displacement*). Gdy specyfikator trybu adresowania wskazuje, że jest używane przesunięcie, dodawane jest pole przesunięcia zawierające 8-, 16- lub 32-bitową liczbę całkowitą ze znakiem.
- **Natychmiastowe** (*immediate*). Zawiera wartość 8-, 16- lub 32-bitowego argumentu.

Pożyteczne może być teraz przeprowadzenie kilku porównań. W formacie Pentium tryb adresowania stanowi raczej część sekwencji kodu operacji niż argumentu. Ponieważ tylko jeden argument może nieść informację o trybie adresowania, w rozkazie może występować tylko jedno odniesienie do argumentu w pamięci. W przeciwieństwie do tego VAX przenosi informację o trybie adresowania z każdym argumentem, co pozwala na operacje pamięć-do-pamięci. Rozkazy Pentium są bardziej zwarte. Jeśli jednak jest wymagana operacja pamięć-do-pamięci, VAX może ją uruchomić za pomocą jednego rozkazu.

Format Pentium pozwala na użycie przy indeksowaniu adresów względnych nie ograniczających się do przesunięć 1-bajtowych, lecz również umożliwia adresy o przesunięciach 2- lub 4-bajtowych dla potrzeb indeksowania. Chociaż zastosowanie długiego przesunięcia indeksowego powoduje wydłużenie rozkazu, cecha ta pozwala na uzyskanie potrzebnej elastyczności. Jest ona na przykład użyteczna przy adresowaniu dużych tablic lub ramek stosów. W przeciwieństwie do tego, format rozkazu IBM S/370 umożliwia adresowanie lokalne nie więcej niż 4 KB (12-bitowa informacja o przesunięciu), a przesunięcie musi być dodatnie. Gdy lokacja znajduje się poza zasięgiem tego adresu względnego, kompilator musi generować dodatkowy kod w celu uzyskania potrzebnego adresu. Problem ten jest szczególnie widoczny podczas operowania ramkami stosu, które mają zmienne lokalne zajmujące ponad 4 KB. Jak stwierdzono w [DEWA90], „generowanie kodów dla 370 jest tak bolesne, że istniały nawet kompilatory dla tych maszyn, które po prostu ograniczały rozmiar ramki stosu do 4 KB".

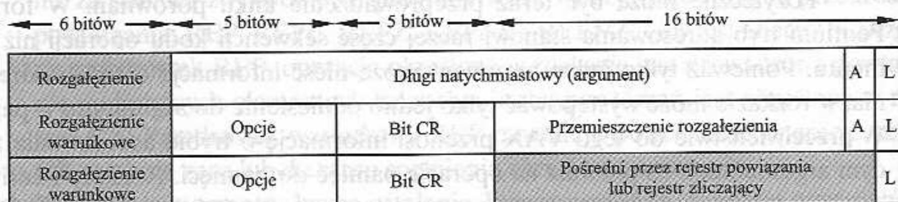
Jak można zauważyć, dekodowanie listy rozkazów Pentium jest bardzo skomplikowane. Wynikło to częściowo z potrzeby wstecznej kompatybilności z maszyną 8086, a częściowo z dążenia części projektantów do zapewnienia piszącym kompilatory wszelkiej możliwej pomocy w tworzeniu efektywnych kodów. Jest sprawą do dyskusji, czy tak skomplikowana lista rozkazów może stanowić lepszy wybór w stosunku do znajdującej się na przeciwnym biegunie listy rozkazów RISC.

Formaty rozkazów PowerPC

Wszystkie rozkazy procesora PowerPC mają długość 32 bity i odpowiadają pewnemu regularnemu formatowi. Pierwsze 6 bitów rozkazu określa operację, która ma być wykonana. W pewnych przypadkach w innej części rozkazu znajduje się rozszerzenie kodu operacji, które określa szczególny przypadek operacji. Na rysunku 11.9 bity kodu operacji są reprezentowane przez zacieniowane obszary każdego z formatów.

Zwróćmy uwagę na regularną strukturę formatów, która ułatwia pracę jednostkom wykonującym rozkazy. W przypadku wszystkich rozkazów arytmetycznych, logicznych oraz ładuj/zapisz po kodzie operacji występują dwa 5-bitowe odniesienia do rejestru, co pozwala na używanie 32 rejestrów roboczych.

Rozkazy rozgałęzienia zawierają bit powiązania (L), który wskazuje, że efektywny adres rozkazu następującego po rozkazie rozgałęzienia ma być umieszczony w rejestrze powiązania. Dwie postacie rozkazów zawierają także bit (A) wskazujący,



(a) Rozkazy rozgałęzienia

CR	Bit miejsca przeznaczenia	Bit źródła	Bit źródła	Dodawanie, OR, XOR itd.	/

(b) Rozkazy logiczne rejestru warunkowego

Pośrednie ładowanie/zapis	Rejestr przeznaczenia	Rejestr podstawowy	Przesunięcie		
Pośrednie ładowanie/zapis	Rejestr przeznaczenia	Rejestr podstawowy	Rejestr indeksowy	Rozmiar, znak, aktualizacja	/
Pośrednie ładowanie/zapis	Rejestr przeznaczenia	Rejestr podstawowy	Przesunięcie	XO	*

(c) Rozkazy ładowania/zapisu

Arytmetyczny	Rejestr przeznaczenia	Rejestr źródłowy	Rejestr źródłowy	O	Dodawanie, odejmowanie itd.		R	
Dodawanie, odejmowanie itd.	Rejestr przeznaczenia	Rejestr źródłowy	Wartość natychmiastowa ze znakiem					
Logiczny	Rejestr źródłowy	Rejestr przeznaczenia	Rejestr źródłowy	Dodawanie, OR, XOR itd.			R	
AND, OR itd.	Rejestr źródłowy	Rejestr przeznaczenia	Wartość natychmiastowa bez znaku					
Rotacja	Rejestr źródłowy	Rejestr przeznaczenia	Rejestr źródłowy	Początek maski	Koniec maski		R	
Rotacja lub przesunięcie	Rejestr źródłowy	Rejestr przeznaczenia	Rejestr źródłowy	Rodzaj przesunięcia lub maska			R	
Rotacja	Rejestr źródłowy	Rejestr przeznaczenia	Rejestr źródłowy	Maska	XO	S	R	*
Rotacja	Rejestr źródłowy	Rejestr przeznaczenia	Rejestr źródłowy	Maska	XO		R	*
Przesunięcie	Rejestr źródłowy	Rejestr przeznaczenia	Wielkość przesunięcia	Rodzaj przesunięcia lub maska		S	R	*

(d) Rozkazy arytmetyczne całkowitoliczbowe, logiczne oraz przesunięcia i rotacji

Zmiennopozycyjna pojedyncza/podwójna	Rejestr przeznaczenia	Rejestr źródłowy	Rejestr źródłowy	Rejestr źródłowy	Dodawanie zmiennopozycyjne itd.	R

(e) Rozkazy arytmetyczne zmiennopozycyjne

A – bezwzględna lub w stosunku do PC
L – łącze do podprogramu
O – przepełnienie rekordu w XER
R – warunki rekordu w CR1
XO – rozszerzenie kodu operacji
S – część pola wielkości przesunięcia
* – tylko w implementacjach 64-bitowych

Rysunek 11.9. Formaty rozkazu procesora PowerPC

czy tryb adresowania jest bezwzględny, czy zależny od PC (licznika programu). W przypadku warunkowych rozkazów rozgałęzienia pole bitowe CR określa bit, który ma być testowany w rejestrze warunku. Pole opcji określa warunki dokonania rozgałęzienia. Mogą być ustalone następujące warunki:

- rozgałęziaj zawsze;
- rozgałęziaj, gdy wynik obliczeń jest różny od 0 i warunek jest fałszywy (niespełniony);
- rozgałęziaj, gdy wynik obliczeń jest różny od 0 i warunek jest prawdziwy (spełniony);
- rozgałęziaj, gdy wynik obliczeń jest równy 0 i warunek jest fałszywy;
- rozgałęziaj, gdy wynik obliczeń jest równy 0 i warunek jest prawdziwy;
- rozgałęziaj, gdy wynik obliczeń jest różny od 0;
- rozgałęziaj, gdy wynik obliczeń jest równy 0;
- rozgałęziaj, gdy warunek jest fałszywy;
- rozgałęziaj, gdy warunek jest prawdziwy.

Większość rozkazów, których wynikiem są obliczenia (arytmetyczne, arytmetyczne zmiennopozycyjne, logiczne), zawiera bit wskazujący, czy wynik operacji powinien być zapisany w rejestrze warunku. Jak pokażemy, cecha ta jest użyteczna do przetwarzania z przewidywaniem rozgałęzienia.

Rozkazy zmiennopozycyjne mają pola dla trzech rejestrów źródłowych. W wielu przypadkach są używane tylko dwa rejestry źródłowe. Kilka rozkazów obejmuje mnożenie dwóch rejestrów źródłowych, a następnie dodanie lub odjęcie trzeciego rejestru źródłowego. Włączono te złożone rozkazy, kierując się częstością ich stosowania. Na przykład iloczyn wewnętrzny będący częścią wielu operacji macierzowych może być zaimplementowany za pomocą mnożenia-dodawania.

11.5. Polecana literatura

Pozycje literatury wymienione w rozdziale 10 są równie przydatne do tego rozdziału. [BLAA97] zawiera szczegółowe omówienie formatów rozkazów i trybów adresowania. Ponadto można sięgnąć do [FLYN85], gdzie znajduje się analiza problemów projektowania list rozkazów, szczególnie dotyczących formatu rozkazu.

BLAA97 Blaauw G., Brooks F.: *Computer Architecture: Concepts and Evolution*. Reading, Addison-Wesley, 1997.

FLYN85 Flynn M., Johnson J., Wakefield S.: „On instruction Sets and Their Formats". *IEEE Trans. on Computers*, March 1985.

11.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Adres efektywny – *effective address*
Adresowanie bezpośrednie – *direct addressing*
Adresowanie natychmiastowe – *immediate addressing*
Adresowanie pośrednie – *indirect addressing*
Adresowanie rejestrowe – *register addressing*
Adresowanie rejestrowe pośrednie – *register indirect addressing*
Adresowanie względne – *relative addressing*
Adresowanie z przesunięciem – *displacement addressing*
Adresowanie z rejestrem podstawowym – *base-register addressing*
Autoindeksowanie – *autoindexing*
Format rozkazu – *instruction format*
Indeksowanie – *indexing*
Indeksowanie wstępne – *preindexing*
Indeksowanie wtórne – *postindexing*
Słowo – *word*

Pytania kontrolne

11.1. Krótko zdefiniuj adresowanie natychmiastowe.
11.2. Krótko zdefiniuj adresowanie bezpośrednie.
11.3. Krótko zdefiniuj adresowanie pośrednie.
11.4. Krótko zdefiniuj adresowanie rejestrowe.
11.5. Krótko zdefiniuj adresowanie rejestrowe pośrednie.
11.6. Krótko zdefiniuj adresowanie z przesunięciem.
11.7. Krótko zdefiniuj adresowanie względne.
11.8. Jakie korzyści wynikają z autoindeksowania?
11.9. Jaka jest różnica między indeksowaniem wtórnym a wstępnym?
11.10. Jakie fakty muszą być uwzględniane przy określaniu bitów adresowania w rozkazie?
11.11. Jakie są wady i zalety posługiwania się formatem rozkazu o zmiennej długości?

Problemy do rozwiązania

11.1. Uzasadnij twierdzenie, że prawdopodobnie rozkaz 32-bitowy jest o wiele mniej niż 2-krotnie bardziej użyteczny w porównaniu z rozkazem 16-bitowym.
11.2. Dane są następujące wartości pamięci i maszyna 1-adresowa z akumulatorem. Jakie wartości zostaną załadowane do akumulatora na podstawie następujących rozkazów?

- słowo 20 zawiera 40;
- słowo 30 zawiera 50;
- słowo 40 zawiera 60;
- słowo 50 zawiera 70.

(a) LOAD IMMEDIATE 20 (ładuj natychmiastowo 20)
(b) LOAD DIRECT 20 (ładuj bezpośrednio 20)
(c) LOAD INDIRECT 20 (ładuj pośrednio 20)
(d) LOAD IMMEDIATE 30 (ładuj natychmiastowo 30)

(e) LOAD DIRECT 30 (ładuj bezpośrednio 30)
(f) LOAD INDIRECT 30 (ładuj pośrednio 30)*

11.3. Niech adres przechowywany w liczniku programu będzie oznaczony symbolem X1. Rozkaz przechowywany w X1 ma część adresową (odniesienie do argumentu) X2. Argument potrzebny do wykonania rozkazu jest przechowywany w słowie pamięci o adresie X3. Rejestr indeksu zawiera wartość X4. Jakie są wzajemne związki między tymi wielkościami, jeśli tryb adresowania jest:

(a) bezpośredni,
(b) pośredni,
(c) względny wobec licznika PC,
(d) indeksowany?

11.4. Pole adresu w rozkazie zawiera wartość dziesiętną 14. Gdzie znajduje się odpowiedni argument w przypadku:

(a) adresowania natychmiastowego?
(b) adresowania bezpośredniego?
(c) adresowania pośredniego?
(d) adresowania rejestrowego?
(e) adresowania rejestrowego pośredniego?

11.5. Rozkaz rozgałęzienia sformułowany w trybie adresowania względnego wobec licznika PC jest przechowywany w pamięci pod adresem 620_{10}. Skok jest dokonywany do lokacji 530_{10}. Pole adresowe rozkazu ma długość 10 bitów. Jaka wartość binarna jest w rozkazie?

11.6. Ile razy procesor musi się odwoływać do pamięci, gdy pobiera i wykonuje rozkaz w trybie adresowania pośredniego, jeśli rozkaz jest: (a) obliczeniem wymagającym jednego argumentu; (b) rozgałęzieniem?

11.7. IBM 370 nie przewiduje adresowania pośredniego. Załóżmy, że adres argumentu znajduje się w pamięci głównej. Jak można uzyskać dostęp do tego argumentu?

11.8. Dlaczego decyzja IBM przejścia od słowa 36-bitowego do 32-bitowego była bolesna i dla kogo?

11.9. W pracy [COOK82] autor proponuje, żeby tryby adresowania względnego wobec licznika PC zostały wyeliminowane na rzecz innych trybów, takich jak użycie stosu. Jaka jest wada tej propozycji?

11.10. Załóżmy, że lista rozkazów przewiduje używanie ustalonej, 16-bitowej długości rozkazu. Specyfikatory argumentu mają długość 6 bitów. W liście znajduje się K rozkazów 2-argumentowych i L 0-argumentowych. Ile maksymalnie może być rozkazów 1-argumentowych?

11.11. Zaprojektuj kod operacji o zmiennej długości umożliwiający zakodowanie poniższych rozkazów w postaci rozkazów 36-bitowych:

- rozkazów z dwoma adresami 15-bitowymi i jednym 3-bitowym numerem rejestru;
- rozkazów z jednym adresem 15-bitowym i jednym 3-bitowym numerem rejestru;
- rozkazów bez adresów i rejestrów.

11.12. Rozważ wyniki problemu 10.3. Załóż, że M jest 16-bitowym adresem pamięci i że X, Y i Z są albo 16-bitowymi adresami, albo 4-bitowymi numerami rejestrów. Maszyna 1-adresowa używa akumulatora, natomiast maszyny 2- i 3-adresowe mają 16 rejestrów

* Określenia towarzyszące rozkazowi LOAD (ładuj) dotyczą trybu adresowania (*przyp. tłum.*).

oraz rozkazy operujące na wszystkich kombinacjach lokacji pamięci i rejestrów. Załóż 8-bitowy kod operacji oraz długości rozkazu będące wielokrotnością 4 bitów. Ilu bitów potrzebuje każda z maszyn w celu obliczenia X?

11.13. Czy istnieje możliwe do przyjęcia uzasadnienie stosowania rozkazów z dwoma kodami operacji?

11.14. Lista rozkazów procesora Pentium obejmuje następujący rozkaz:

```
IMUL op1, op2, immediate
```

Rozkaz ten powoduje pomnożenie wartości op2, która może być zawarta w rejestrze lub w pamięci, przez natychmiastową wartość argumentu, i umieszczenie wyniku w op1, który musi być rejestrem. W liście rozkazów nie występują inne 3-argumentowe rozkazy tego rodzaju. Jakie jest możliwe zastosowanie tego rozkazu? *Wskazówka*: rozważ indeksowanie.

Rozdział 12

Struktura i działanie procesora

12.1. Organizacja procesora
12.2. Organizacja rejestrów
Rejestry widzialne dla użytkownika
Rejestry sterowania i stanu
Przykładowe organizacje rejestrów w mikroprocesorach
12.3. Cykl rozkazu
Cykl pośredni
Przepływ danych
12.4. Potokowe przetwarzanie rozkazów
Strategia przetwarzania potokowego
Wydajność przetwarzania potokowego
Postępowanie z rozgałęzieniami
Przetwarzanie potokowe w procesorze Intel 80486
12.5. Procesor Pentium
Organizacja rejestrów
Przetwarzanie przerwań
12.6. Procesor PowerPC
Organizacja rejestrów
Przetwarzanie przerwań
12.7. Polecana literatura
12.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

PODSTAWOWE SPOSTRZEŻENIA

- Procesor zawiera zarówno rejestry widzialne dla użytkownika, jak i rejestry sterowania i stanu. Do tych pierwszych można się odwoływać – wprost lub domyślnie – w rozkazach maszynowych. Rejestry widzialne dla użytkownika mogą mieć przeznaczenie ogólne albo mogą być wyspecjalizowane, np. służyć do zapisywania liczb stałopozycyjnych lub zmiennopozycyjnych, adresów, indeksów i wskaźników segmentów. Rejestry sterowania i stanu służą do sterowania pracą procesora. Jednym z oczywistych przykładów jest licznik rozkazów. Innym ważnym przykładem jest słowo stanu programu (*program status word* – PSW), które zawiera różne bity określające stan i warunki. Należą do nich bity odzwierciedlające wynik ostatnich operacji arytmetycznych, bity zezwolenia przerwania oraz wskaźnik pracy procesora w trybie nadzorcy lub użytkownika.
- W celu przyspieszenia wykonywania programów stosuje się w procesorach potokowe przetwarzanie rozkazów. W gruncie rzeczy przetwarzanie potokowe polega na podziale cyklu rozkazu na pewną liczbę oddzielnych etapów realizowanych szeregowo, takich jak pobieranie rozkazu, dekodowanie go, określanie adresu argumentu, pobieranie argumentu, wykonywanie rozkazu i zapisanie argumentu wynikowego. Rozkazy przechodzą przez te etapy podobnie jak na linii montażowej, dzięki czemu w zasadzie jednocześnie każdy etap może być realizowany w odniesieniu do innego rozkazu. Występowanie rozgałęzień i zależności między rozkazami komplikuje projektowanie i stosowanie potoków.

W tym rozdziale zajmiemy się tymi aspektami procesora, które nie zostały ujęte w części trzeciej. Stworzymy też podstawy do dyskusji na temat komputerów RISC oraz architektury superskalarnej w rozdz. 13 i 14.

Rozpoczniemy od podsumowania organizacji procesora. Opiszemy rejestry, które stanowią wewnętrzną pamięć procesora. Następnie wrócimy do dyskusji (rozpoczętej w podrozdz. 3.2) na temat cyklu rozkazu. Omówimy cykl rozkazu oraz powszechnie stosowaną metodę przetwarzania potokowego. Na zakończenie przeanalizujemy pewne dodatkowe aspekty organizacji Pentium i PowerPC.

12.1. Organizacja procesora

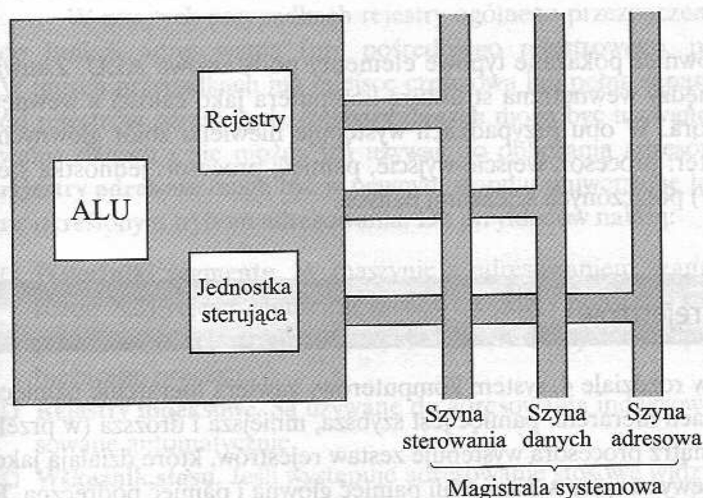
Aby zrozumieć organizację procesora, rozważmy stawiane mu wymagania, a więc zadania, które musi on realizować:

- **Pobieranie rozkazów.** Procesor musi odczytywać rozkazy z pamięci.
- **Interpretowanie rozkazów.** Rozkazy muszą być zdekodowane w celu określenia, jakie działania są wymagane.

- **Pobieranie danych.** Wykonywanie rozkazów może wymagać odczytywania danych z pamięci lub z modułu wejścia-wyjścia.
- **Przetwarzanie danych.** Wykonywanie rozkazów może wymagać przeprowadzenia na danych pewnych operacji arytmetycznych lub logicznych.
- **Zapisanie danych.** Wyniki operacji mogą wymagać zapisania danych w pamięci lub w module wejścia-wyjścia.

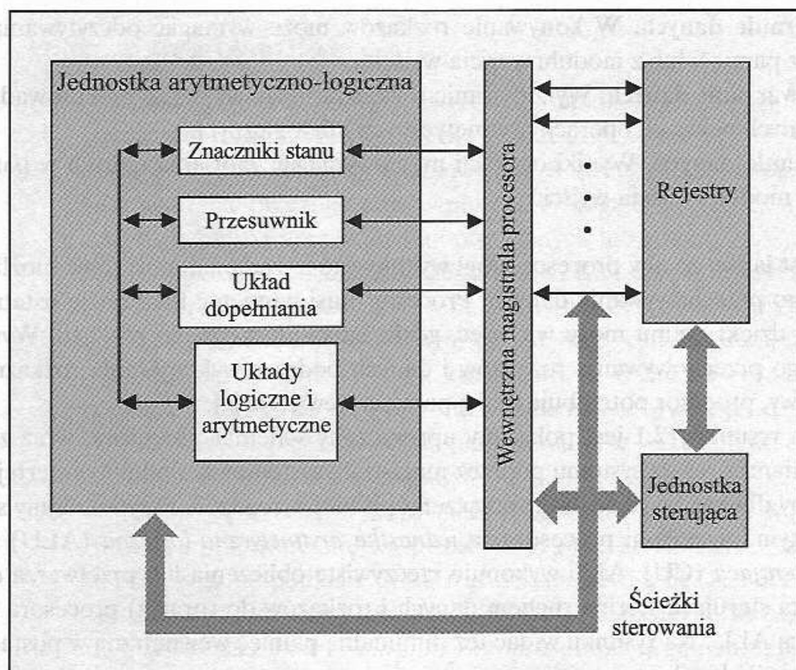
Jest jasne, że aby procesor mógł wykonywać te zadania, musi mieć możliwość czasowego przechowywania danych. Procesor musi pamiętać lokalizację ostatniego rozkazu, dzięki czemu może wiedzieć, gdzie szukać następnego rozkazu. Wymaga czasowego przechowywania rozkazów i danych podczas wykonywania rozkazu. Innymi słowy, procesor potrzebuje małej pamięci wewnętrznej.

Na rysunku 12.1 jest pokazany uproszczony schemat procesora, wraz z jego połączeniami z resztą systemu poprzez magistralę systemową. Podobny interfejs jest potrzebny dla każdej ze struktur połączeń opisanej w rozdz. 3. Przypomnijmy sobie, że głównymi zespołami procesora są *jednostka arytmetyczno-logiczna* (ALU) i *jednostka sterująca* (CU). ALU wykonuje rzeczywiste obliczenia lub przetwarza dane. Jednostka sterująca steruje ruchem danych i rozkazów do (oraz z) procesora i steruje pracą ALU. Na rysunku widać też minimalną pamięć wewnętrzną w postaci zespołu lokacji określanego mianem *rejestrów*.



Rysunek 12.1. Procesor z magistralą systemową

Na rysunku 12.2 widać nieco bardziej szczegółowy schemat procesora. Są na nim pokazane ścieżki transferu danych i sterowania logicznego, włącznie z elementem określonym jako **wewnętrzna magistrala procesora**. Element ten jest potrzebny do przesyłania danych między różnymi rejestrami a ALU, ponieważ w rzeczywistości ALU operuje tylko na danych pochodzących z wewnętrznej pamięci procesora.



Rysunek 12.2. Wewnętrzna struktura procesora

Na rysunku są również pokazane typowe elementy podstawowe ALU. Zauważmy podobieństwo między wewnętrzną strukturą komputera jako całości a wewnętrzną strukturą procesora. W obu przypadkach występuje niewielki zbiór głównych elementów (komputer: procesor, wejście-wyjście, pamięć; procesor: jednostka sterująca, ALU, rejestry) połączonych ścieżkami danych.

12.2. Organizacja rejestrów

Jak omówiliśmy w rozdziale 4, system komputerowy zawiera hierarchię pamięci. Na wyższych poziomach hierarchii pamięć jest szybsza, mniejsza i droższa (w przeliczeniu na bit). Wewnątrz procesora występuje zestaw rejestrów, które działają jako poziom pamięci przewyższający w hierarchii pamięć główną i pamięć podręczną. Rejestry w procesorze można podzielić na dwie grupy:

- **Rejestry widzialne dla użytkownika.** Umożliwiają osobie programującej w języku maszynowym lub symbolicznym minimalizowanie odniesień do pamięci głównej drogą optymalizowania wykorzystania rejestrów.
- **Rejestry sterowania i stanu.** Są one używane przez jednostkę sterującą do sterowania pracą procesora oraz przez uprzywilejowane programy systemu operacyjnego do sterowania wykonywaniem programów.

Nie ma ścisłego rozgraniczenia między tymi dwiema kategoriami. Na przykład w pewnych maszynach licznik programu jest widzialny dla użytkownika (np. Pentium), ale w wielu tak nie jest (np. PowerPC). Jednak dla potrzeb dalszej dyskusji będziemy posługiwali się tymi kategoriami.

Rejestry widzialne dla użytkownika

Rejestr widzialny dla użytkownika jest rejestrem, do którego można się odnosić za pomocą języka maszynowego, którym posługuje się procesor. Możemy je scharakteryzować, posługując się następującym podziałem:

- ogólnego przeznaczenia, czyli robocze (*general purpose*),
- danych,
- adresów,
- kodów warunkowych.

Rejestry ogólnego przeznaczenia (robocze) mogą być przypisane przez programistę wielu funkcjom. Czasem ich użycie w ramach listy rozkazów ma charakter ortogonalny w stosunku do operacji. Znaczy to, że dowolny rejestr roboczy może zawierać argument dowolnego kodu operacji. Jest to prawdziwe zastosowanie robocze (uniwersalne). Często jednak występują ograniczenia. Mogą na przykład występować rejestry dedykowane operacjom zmiennopozycyjnym.

W pewnych przypadkach rejestry ogólnego przeznaczenia mogą być używane do funkcji adresowania (np. pośredniego rejestrowego, przemieszczeniowego). W innych przypadkach ma miejsce częściowa lub pełna separacja rejestrów danych od rejestrów adresowych. **Rejestry danych** mogą być używane tylko do przechowywania danych i nie można ich używać do obliczania adresów argumentów. Same **rejestry adresowe** mogą być w pewnym stopniu uniwersalne lub mogą być przypisane określonym trybom adresowania. Do przykładów należą:

- ❑ **Wskaźniki segmentu.** W maszynie z adresowaniem segmentowym (patrz podrozdz. 8.3) rejestr segmentu zachowuje adres podstawy segmentu. Może występować wiele rejestrów: na przykład jeden dla systemu operacyjnego i jeden dla bieżącego procesu.
- ❑ **Rejestry indeksowe.** Są używane do adresowania indeksowego i mogą być indeksowane automatycznie.
- ❑ **Wskaźnik stosu.** Jeśli występuje adresowanie stosowe widzialne dla użytkownika, to zwykle stos znajduje się w pamięci i jest używany dedykowany rejestr wskazujący wierzchołek stosu. Pozwala to na adresowanie domyślne; to znaczy, że rozkazy umieszczania na stosie, zdejmowania ze stosu i inne rozkazy związane ze stosem nie muszą zawierać jawnego adresu argumentu.

Występuje tu kilka problemów projektowych. Ważnym z nich jest to, czy zastosować wyłącznie uniwersalne rejestry robocze, czy wyspecjalizowane. Zetknęliśmy się już z tym zagadnieniem w poprzednim rozdziale, ponieważ ma ono wpływ na projektowanie list rozkazów. Przy stosowaniu rejestrów wyspecjalizowanych

z kodu operacji może domyślnie wynikać, do którego rodzaju rejestru odnosi się określony specyfikator argumentu. Specyfikator argumentu musi tylko określić jeden z zespołu rejestrów wyspecjalizowanych, a nie jeden ze wszystkich rejestrów, co pozwala na zaoszczędzenie bitów. Z drugiej strony specjalizacja ogranicza elastyczność programistyczną.

Innym zagadnieniem projektowym jest liczba rejestrów (ogólnego przeznaczenia lub wyspecjalizowanych). Jak poprzednio, wpływa to na projekt listy rozkazów, ponieważ więcej rejestrów oznacza więcej bitów specyfikatora argumentu. Jak już wspomnieliśmy, optimum wydaje się leżeć między 8 a 32 rejestrami [LUND77]. Mniej rejestrów oznacza więcej odniesień do pamięci; z kolei więcej rejestrów nie zmniejsza zauważalnie ilości odniesień do pamięci (patrz np. [WILL90]). Jednak w systemach RISC wystąpiło nowe podejście, oparte na stosowaniu setek rejestrów, co opiszemy w rozdz. 13.

Istnieje wreszcie problem długości rejestrów. Rejestry przeznaczone do przechowywania adresów muszą oczywiście mieć długość wystarczającą do przechowywania najdłuższego adresu. Rejestry danych powinny móc przechowywać większość rodzajów danych. W niektórych maszynach możliwe jest używanie sąsiednich rejestrów do przechowywania wartości o podwójnej długości.

W rejestrach należących do ostatniej kategorii, które są przynajmniej częściowo widzialne dla użytkownika, przechowuje się **kody warunkowe** (nazywane także *znacznikami stanu* lub *flagami*). Kody warunkowe są bitami ustalanymi sprzętowo przez procesor w wyniku operacji. Na przykład operacja arytmetyczna może dać wynik dodatni, ujemny, zerowy lub przepełnienie. Poza skierowaniem do rejestru lub do pamięci samego wyniku, jest ustalany również kod warunkowy. Kod ten może być następnie sprawdzany w ramach operacji rozgałęzienia warunkowego.

Bity kodu warunkowego są zbierane w jednym lub w wielu rejestrach. Zwykle stanowią one część zawartości rejestru sterowania. Na ogół rozkazy maszynowe umożliwiają odczytywanie tych bitów za pomocą odniesienia domyślnego, jednak nie mogą one być zmieniane przez programistę.

W niektórych maszynach wywołanie podprogramu powoduje automatyczne zapisywanie zawartości wszystkich rejestrów widzialnych dla użytkownika, co umożliwia ich odtworzenie po powrocie z podprogramu. Zapisywanie i odtwarzanie są wykonywane przez procesor w ramach wykonywania rozkazów wywołania i powrotu. Dzięki temu każdy podprogram może niezależnie używać rejestrów widzialnych dla użytkownika. W innych maszynach obowiązkiem programisty jest zapisanie zawartości odpowiednich rejestrów widzialnych dla użytkownika przed wywołaniem podprogramu, przez włączenie do programu niezbędnych rozkazów.

Rejestry sterowania i stanu

W procesorze występują różnorodne rejestry wykorzystywane do sterowania jego pracą. Większość z nich, w większości maszyn, jest niewidzialna dla użytkownika. Niektóre z nich mogą być widzialne dla rozkazów maszynowych wykonywanych w trybie sterowania lub przez system operacyjny.

Oczywiście różne maszyny mają różne organizacje rejestrów; jest też stosowana różna terminologia. Przedstawimy rozsądnie kompletną listę rodzajów rejestrów wraz z krótkim opisem.

Cztery rejestry mają zasadnicze znaczenie dla wykonywania rozkazów:

- **Licznik programu (PC).** Zawiera adres rozkazu przewidzianego do pobrania.
- **Rejestr rozkazu (IR).** Zawiera ostatnio pobrany rozkaz.
- **Rejestr adresowy pamięci (MAR).** Zawiera adres lokacji w pamięci.
- **Rejestr buforowy pamięci (MBR).** Zawiera słowo danych, które ma być zapisane do pamięci lub które zostało ostatnio odczytane z pamięci.

Zwykle licznik programu jest aktualizowany przez procesor po każdym pobraniu rozkazu, dzięki czemu zawsze wskazuje on następny rozkaz przewidziany do pobrania. Rozkaz rozgałęzienia lub pominięcia również modyfikuje zawartość licznika PC. Pobrany rozkaz jest ładowany do rejestru rozkazu, gdzie analizowane są kod operacji i specyfikator argumentu. Dane są wymieniane z pamięcią za pośrednictwem rejestrów MAR i MBR. W systemie magistralowym rejestr MAR jest połączony bezpośrednio z magistralą adresową, a MBR – z magistralą danych. Z kolei rejestry widzialne dla użytkownika wymieniają dane z rejestrem MBR.

Cztery wspomniane rejestry są używane do przenoszenia danych między procesorem a pamięcią. Wewnątrz procesora dane muszą być doprowadzone do ALU w celu ich przetwarzania. ALU musi mieć bezpośredni dostęp do rejestru MBR i rejestrów widzialnych dla użytkownika. Alternatywnie mogą występować dodatkowe rejestry buforujące na granicy z ALU; rejestry te służą jako rejestry wejściowe oraz wyjściowe dla ALU i wymieniają dane z rejestrem MBR i z rejestrami widzialnymi dla użytkownika.

We wszystkich projektach procesora występuje rejestr lub zespół rejestrów, określanych często jako *słowo stanu programu* (*program status word* – PSW). Rejestry te zawierają informację o stanie. W rejestrach PSW są zwykle przechowywane kody warunkowe i inne informacje o stanie. Występują tam zwykle następujące pola (znaczniki stanu, czyli flagi):

- **Znak.** Zawiera bit znaku wyniku ostatniej operacji arytmetycznej.
- **Zero.** Ustawiane, gdy wynik jest równy zeru.
- **Przeniesienie.** Ustawiane, gdy wynikiem operacji jest przeniesienie (dodawanie) lub przeniesienie zanegowane (odejmowanie). Używane przy operacjach arytmetycznych obejmujących wiele słów.
- **Równość.** Ustawiane, gdy wynikiem porównywania logicznego jest równość.
- **Przepełnienie.** Używane do wskazywania przepełnienia arytmetycznego.
- **Zezwolenie/blokowanie przerwania.** Używane do obsługi przerwań.
- **Nadzorca.** Wskazuje, czy procesor pracuje w trybie nadzorcy, czy w trybie użytkownika. Niektóre rozkazy uprzywilejowane mogą być wykonywane tylko w trybie nadzorcy. Również pewne obszary pamięci mogą być dostępne tylko w trybie nadzorcy.

Istnieje wiele innych rejestrów związanych ze stanem i ze sterowaniem, które można odnaleźć w konkretnych projektach procesorów. Obok PSW może występo-

wać wskaźnik bloku pamięci zawierającego dodatkowe informacje o stanie (np. bloków sterowania procesem). W maszynach stosujących przerwania wektorowe może być przewidziany rejestr wektora przerwania. Jeśli do implementacji pewnych funkcji jest używany stos (np. do wywołania podprogramu), to potrzebny jest systemowy wskaźnik stosu. Z systemami pamięci wirtualnej współpracuje wskaźnik tablic stron. Rejestry mogą być wreszcie używane do sterowania operacjami wejścia-wyjścia.

Wiele czynników wpływa na organizację rejestrów sterowania i stanu. Podstawowym zagadnieniem jest wspieranie systemu operacyjnego. Pewne rodzaje informacji sterowania mają szczególne znaczenie dla systemu operacyjnego. Jeśli projektant wie, jak działa przewidziany do stosowania system operacyjny, to organizacja rejestrów może być do pewnego stopnia dostosowana do systemu operacyjnego.

Inną podstawową decyzją projektową jest podział informacji sterowania między rejestry a pamięć. Powszechną praktyką jest przeznaczanie pierwszych (o najniższych adresach) kilkuset lub kilku tysięcy słów pamięci do celów sterowania. Projektant musi zdecydować, ile informacji sterowania musi pozostawać w rejestrach, a ile w pamięci. Zachodzi tu typowa wymienność między kosztem a szybkością.

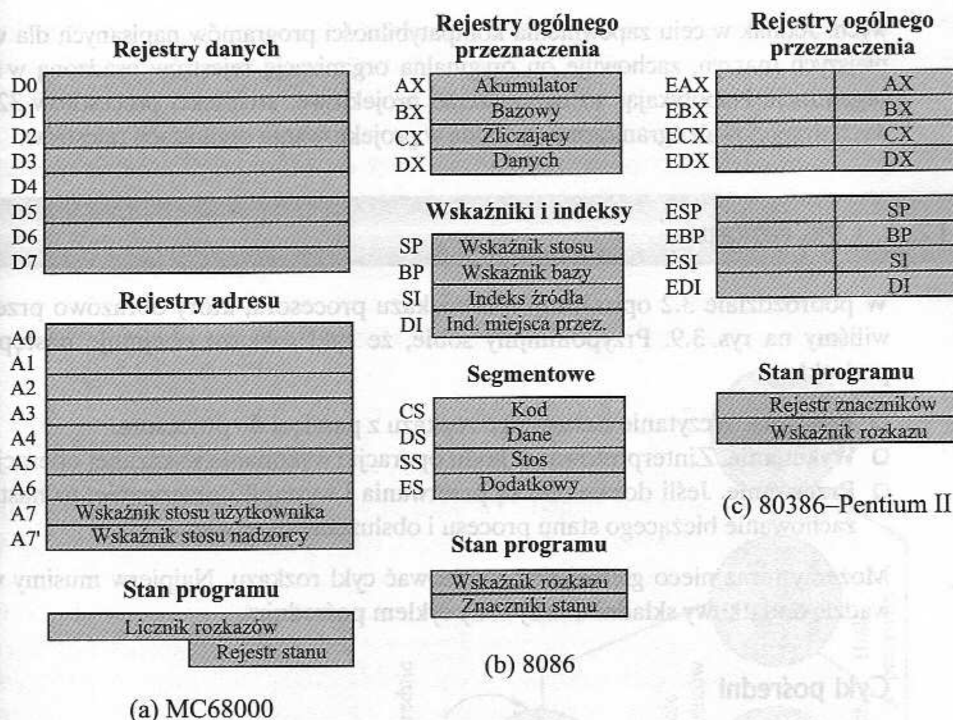
Przykładowe organizacje rejestrów w mikroprocesorach

Pouczające jest przeanalizowanie i porównanie organizacji rejestrów w porównywalnych systemach. W tym punkcie rozpatrzymy dwa mikroprocesory 16-bitowe zaprojektowane prawie w tym samym czasie: Motorola MC68000 [STRI79]oraz Intel 8086 [MORS78]. Na rysunku 12.3a i b widać organizację rejestrów każdego z nich; czysto wewnętrzne rejestry, takie jak rejestry adresu pamięci, nie zostały pokazane.

W procesorze MC68000 rejestry 32-bitowe zostały podzielone na 8 rejestrów danych i 9 rejestrów adresowych. Ośmiu rejestrów danych używa się głównie do manipulowania danymi. Mogą one być również używane do adresowania jako rejestry indeksowe. Wielkość rejestrów umożliwia przeprowadzanie operacji na danych 8-, 16- i 32-bitowych, określonych przez kod operacji. Rejestry adresowe zawierają adresy 32-bitowe (bez segmentowania); dwa z nich są również używane jako wskaźniki stosu, jeden przeznaczony dla użytkowników, a drugi dla systemu operacyjnego, zależnie od bieżącego trybu wykonywania. Oba rejestry noszą numer 7, ponieważ tylko jeden z nich może być używany w określonej chwili. Procesor MC68000 zawiera także 32-bitowy licznik programu i 16-bitowy rejestr stanu.

Zespół Motoroli przyjął bardzo regularną listę rozkazów, bez rejestrów wyspecjalizowanych. Troska o sprawność kodowania doprowadziła do podziału rejestrów na dwie grupy funkcjonalne, co pozwoliło na zaoszczędzenie jednego bitu w każdym specyfikatorze rejestru. Wygląda to na rozsądny kompromis między uniwersalnością a zwartością programów.

W procesorze Intel 8086 przyjęto odmienne podejście do organizacji rejestrów. Każdy rejestr jest wyspecjalizowany, chociaż niektóre z nich mogą również służyć jako rejestry ogólnego przeznaczenia. Procesor 8086 zawiera cztery 16-bitowe rejestry danych, adresowalne na podstawie bajtowej lub 16-bitowej, oraz cztery 16-bitowe rejestry wskaźnikowe i indeksowe. W odniesieniu do niektórych rozkazów rejestry danych



Rysunek 12.3. Przykładowa organizacja rejestrów w mikroprocesorach

mogą służyć jako robocze. W odniesieniu do pozostałych rozkazów są one używane domyślnie. Na przykład rozkaz mnożenia zawsze korzysta z akumulatora. Cztery rejestry wskaźnikowe również w wielu operacjach są używane domyślnie; każdy z nich zawiera adres względny w segmencie. Istnieją również cztery 16-bitowe rejestry segmentu. Trzy z nich są używane w sposób dedykowany (domyślny) w celu wskazania segmentu bieżącego rozkazu (przydatne przy rozkazach rozgałęziania), segmentu zawierającego dane oraz segmentu zawierającego stos. Specjalizacja rejestrów umożliwia uzyskanie zwięzłego kodu kosztem elastyczności. Procesor 8086 ma także wskaźnik rozkazu i zespół 1-bitowych znaczników stanu i sterowania.

Porównanie to powinno być jasne. Jak dotychczas nie istnieje ogólnie akceptowana filozofia dotycząca najlepszego sposobu organizacji rejestrów procesora [TOON81]. Podobnie jak w przypadku projektowania list rozkazów i w przypadku wielu innych problemów projektowania procesorów, jest to wciąż sprawa właściwego osądu i gustu.

Drugi pouczający problem dotyczący projektowania organizacji rejestrów jest zilustrowany na rys. 12.3c. Widać na nim organizację rejestrów widzialnych dla użytkownika w procesorze Intel 80386 [ELAY85], który jest 32-bitowym mikroprocesorem zaprojektowanym jako rozszerzenie 8086[1]. Procesor 80386 używa rejestrów 32-bito-

[1] Ponieważ w procesorze MC68000 zastosowano już rejestry 32-bitowe, w MC68020 [MACG84], stanowiącym pełne rozszerzenie 32-bitowe, użyto tej samej organizacji rejestrów.

wych. Jednak w celu zapewnienia kompatybilności programów napisanych dla wcze-niejszych maszyn, zachowuje on oryginalną organizację rejestrów osadzoną w nowej organizacji. Napotykając to ograniczenie projektowe, architekci procesorów 32-bito-wych dysponowali ograniczoną swobodą w projektowaniu organizacji rejestrów.

12.3. Cykl rozkazu

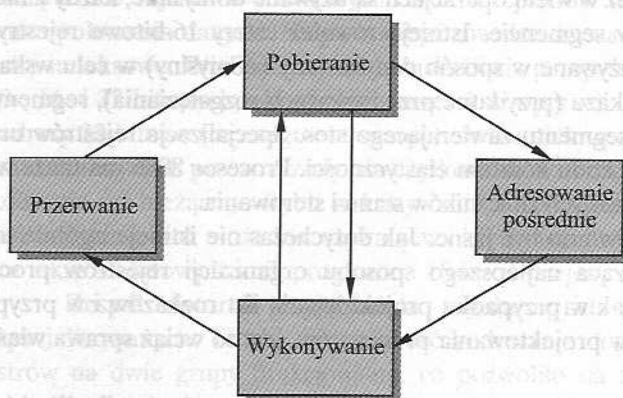
W podrozdziale 3.2 opisaliśmy cykl rozkazu procesora, który obrazowo przedsta-wiliśmy na rys. 3.9. Przypomnijmy sobie, że cykl rozkazu obejmuje następujące podcykle:

- **Pobranie.** Wczytanie następnego rozkazu z pamięci do procesora.
- **Wykonanie.** Zinterpretowanie kodu operacji i wykonanie wskazanej operacji.
- **Przerwanie.** Jeśli dozwolone są przerwania i wystąpiło przerwanie, to następuje zachowanie bieżącego stanu procesu i obsłużenie przerwania.

Możemy teraz nieco głębiej przeanalizować cykl rozkazu. Najpierw musimy wpro-wadzić dodatkowy składnik, nazywany cyklem pośrednim.

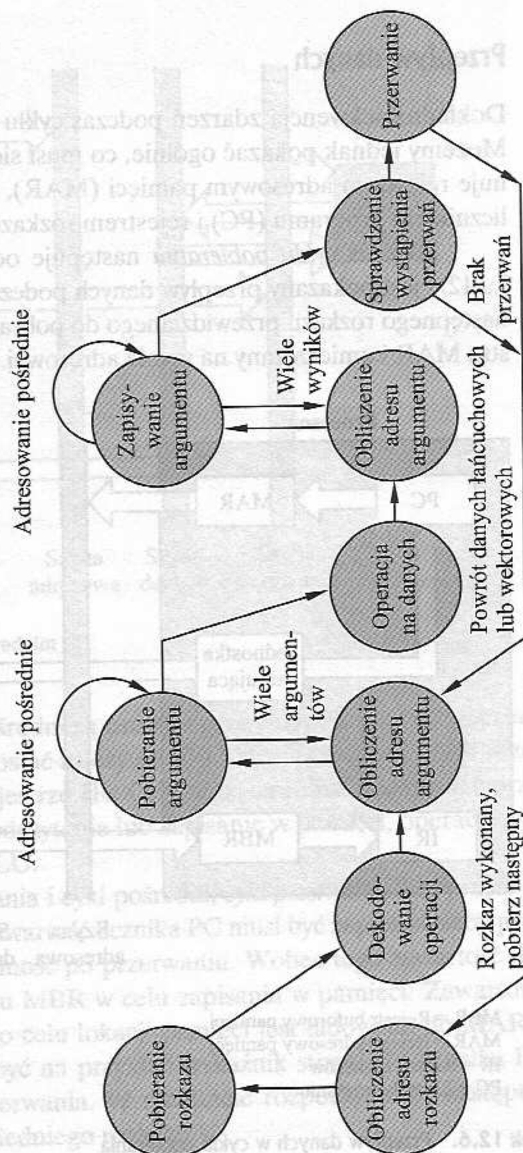
Cykl pośredni

Widzieliśmy w rozdz. 11, że wykonywanie rozkazu może obejmować powołanie na jeden lub więcej argumentów przechowywanych w pamięci, z których każdy wymaga dostępu do pamięci. Ponadto, jeśli jest stosowane adresowanie pośrednie, potrzeb-ne są dodatkowe odniesienia do pamięci.



Rysunek 12.4. Cykl rozkazu

Możemy traktować pobieranie adresów pośrednich jako dodatkowy element cyklu rozkazu. Wynik widać na rys. 12.4. Działanie tego układu polega na przeskaki-waniu między czynnością pobierania rozkazu a czynnością wykonywania rozkazu. Po pobraniu rozkaz jest badany w celu stwierdzenia, czy występuje w nim jakiekolwiek



Rysunek 12.5. Graf stanów cyklu rozkazu

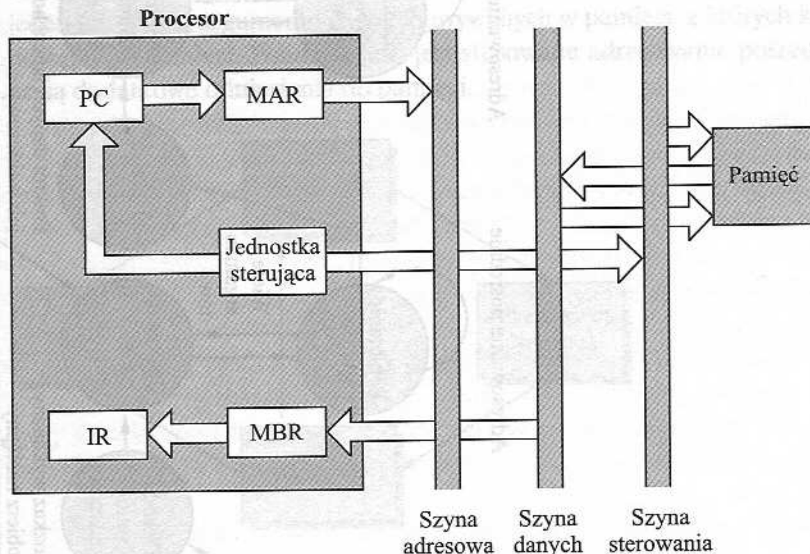
adresowanie pośrednie. Jeśli tak, to wymagane argumenty są pobierane za pomocą adresowania pośredniego. Po wykonaniu może być przetwarzane przerwanie, zanim zostanie pobrany następny rozkaz.

Innym sposobem widzenia tego procesu jest schemat pokazany na rys. 12.5, będący zrewidowaną wersją rys. 3.12. Ilustruje on dokładniej naturę cyklu rozkazu. Gdy rozkaz zostanie pobrany, muszą być zidentyfikowane jego specyfikatory argumentów. Następnie jest pobierany z pamięci każdy argument wejściowy; proces ten może wymagać adresowania pośredniego. Argumenty przechowywane w rejestrach nie muszą być pobierane. Gdy operacja zostanie wykonana, może zajść potrzeba wykonania podobnego ciągu operacji w celu zapisania wyniku w pamięci głównej.

Przepływ danych

Dokładna sekwencja zdarzeń podczas cyklu rozkazu zależy od projektu procesora. Możemy jednak pokazać ogólnie, co musi się zdarzyć. Załóżmy, że procesor dysponuje rejestrem adresowym pamięci (MAR), rejestrem buforowym pamięci (MBR), licznikiem programu (PC) i rejestrem rozkazów (IR).

Podczas *cyklu pobierania* następuje odczytanie adresu z pamięci. Na rysunku 12.6 jest pokazany przepływ danych podczas tego cyklu. Licznik PC zawiera adres następnego rozkazu przewidzianego do pobrania. Adres ten jest przenoszony do rejestru MAR i umieszczany na szynie adresowej. Jednostka sterująca zgłasza zapotrzebo-

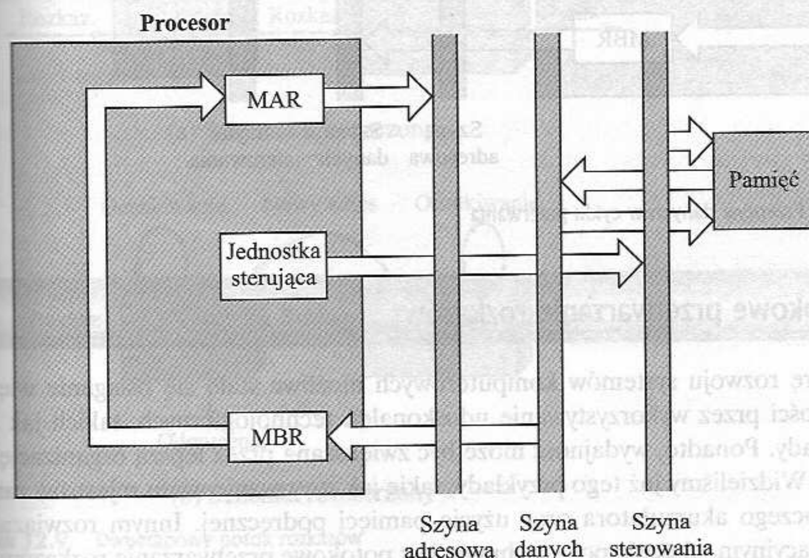


MBR = Rejestr buforowy pamięci
MAR = Rejestr adresowy pamięci
IR = Rejestr rozkazów
PC = Licznik programu

Rysunek 12.6. Przepływ danych w cyklu pobierania

wanie na odczyt z pamięci, a wynik jest umieszczany na szynie danych, kopiowany do MBR, a następnie przenoszony do IR. W tym samym czasie PC przyrasta o 1, co stanowi przygotowanie do pobrania następnego rozkazu.

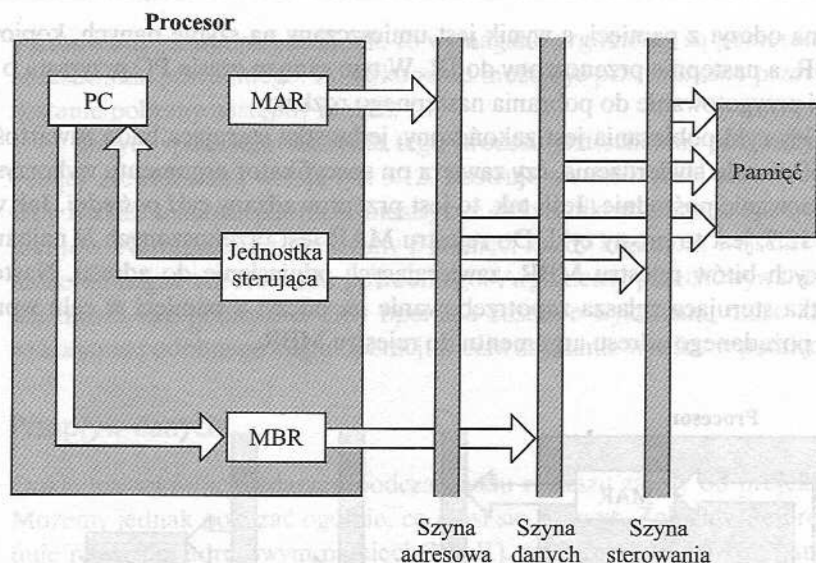
Gdy cykl pobierania jest zakończony, jednostka sterująca bada zawartość rejestru IR w celu stwierdzenia, czy zawiera on specyfikator argumentu wykorzystujący adresowanie pośrednie. Jeśli tak, to jest przeprowadzany *cykl pośredni*. Jak widać na rys. 12.7, jest to prosty cykl. Do rejestru MAR jest przenoszonych *N* najbardziej znaczących bitów rejestru MBR, zawierających odniesienie do adresu. Następnie jednostka sterująca zgłasza zapotrzebowanie na odczyt z pamięci w celu wprowadzenia pożądanego adresu argumentu do rejestru MBR.



Rysunek 12.7. Przepływ danych w cyklu pośrednim

Cykl pobierania i cykl pośredni są proste i przewidywalne. *Cykl rozkazu* przybiera wiele postaci, ponieważ postać zależy od tego, który z różnorodnych rozkazów maszynowych znajduje się w rejestrze IR. Cykl rozkazu może obejmować przesyłanie danych między rejestrami, odczytanie lub zapisanie w pamięci, operację wejścia-wyjścia oraz (lub) wywołanie ALU.

Podobnie jak cykl pobierania i cykl pośredni, *cykl przerwania* jest prosty i przewidywalny (rys. 12.8). Bieżąca zawartość licznika PC musi być zapisana, żeby procesor mógł wznowić normalną działalność po przerwaniu. Wobec tego zawartość licznika PC jest przenoszona do rejestru MBR w celu zapisania w pamięci. Zawartość specjalnej, zarezerwowanej do tego celu lokacji pamięci jest ładowana do MAR przez jednostkę sterującą. Może to być na przykład wskaźnik stosu. Do licznika PC jest ładowany adres procedury przerwania. W rezultacie rozpoczyna się następny cykl rozkazu przez pobranie odpowiedniego rozkazu.



Rysunek 12.8. Przepływ danych w cyklu przerwania

12.4. Potokowe przetwarzanie rozkazów

W miarę rozwoju systemów komputerowych możliwe stało się osiąganie większej wydajności przez wykorzystywanie udoskonaleń technologicznych, takich jak szybsze układy. Ponadto, wydajność może być zwiększana przez lepszą organizację procesora. Widzieliśmy już tego przykłady, takie jak stosowanie wielu rejestrów zamiast pojedynczego akumulatora oraz użycie pamięci podręcznej. Innym rozwiązaniem organizacyjnym, obecnie powszechnym, jest potokowe przetwarzanie rozkazów.

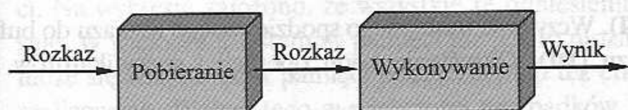
Strategia przetwarzania potokowego

Przetwarzanie potokowe rozkazów jest podobne do użycia linii montażowej w zakładzie produkcyjnym. Dzięki zorganizowaniu procesu produkcyjnego w formie linii montażowej, możliwa jest jednoczesna praca nad wyrobami w różnych stadiach produkcji. Proces ten nazwano *przetwarzaniem potokowym*, ponieważ, tak jak w potoku, na jednym końcu są przyjmowane nowe elementy wejściowe, zanim jeszcze elementy poprzednio przyjęte ukażą się na wyjściu.

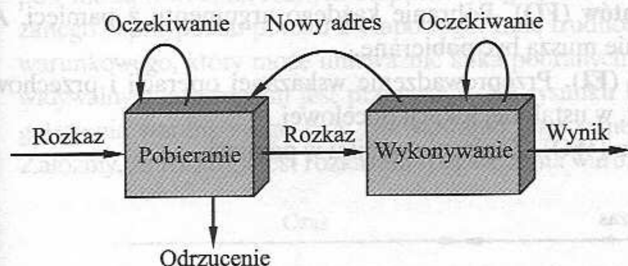
W celu zastosowania tej koncepcji do wykonywania rozkazów musimy zauważyć, że w rzeczywistości wykonywanie rozkazów ma wiele etapów. Na przykład na rys. 12.5 cykl rozkazu został podzielony na 10 kolejno realizowanych zadań. Jest oczywiste, że istnieją tu warunki umożliwiające zastosowanie przetwarzania potokowego.

Rozpatrzmy najpierw prosty podział przetwarzania rozkazu na dwa etapy: pobranie rozkazu i wykonanie rozkazu. Występują takie przedziały czasowe podczas wykonywania rozkazu, w których nie następują odniesienia do pamięci głównej.

Przedziały te mogłyby być wykorzystane do pobrania następnego rozkazu równolegle z wykonywaniem bieżącego. Rozwiązanie to jest przedstawione na rys. 12.9a. Potok przebiega na 2 niezależnych etapach. Na pierwszym etapie następuje pobranie i zbuforowanie rozkazu. Gdy drugi etap jest wolny, następuje przekazanie do niego zbuforowanego rozkazu. Podczas gdy na drugim etapie ma miejsce wykonywanie rozkazu, na pierwszym etapie jest wykorzystywany jeden z nieużywanych cykli pamięci w celu pobrania i zbuforowania następnego rozkazu. Nazywa się to *pobieraniem rozkazu z wyprzedzeniem* (*instruction prefetch*) lub *pobieraniem na zakładkę* (*fetch overlap*).



(a) Schemat uproszczony



(b) Schemat rozszerzony

Rysunek 12.9. Dwuetapowy potok rozkazów

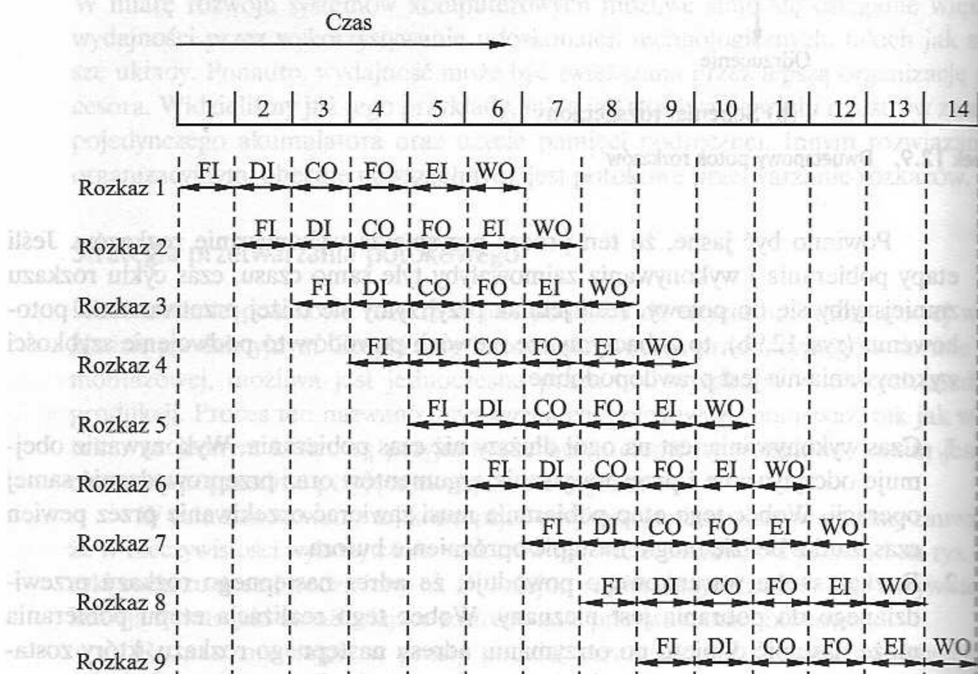
Powinno być jasne, że ten proces przyspiesza wykonywanie rozkazów. Jeśli etapy pobierania i wykonywania zajmowałyby tyle samo czasu, czas cyklu rozkazu zmniejszyłby się do połowy. Jeśli jednak przyjrzymy się bliżej przetwarzaniu potokowemu (rys. 12.9b), to zobaczymy, że z dwóch powodów to podwojenie szybkości wykonywania nie jest prawdopodobne:

1. Czas wykonywania jest na ogół dłuższy niż czas pobierania. Wykonywanie obejmuje odczytywanie i przechowywanie argumentów oraz przeprowadzenie samej operacji. Wobec tego etap pobierania musi zawierać oczekiwanie przez pewien czas, zanim będzie mogło nastąpić opróżnienie bufora.
2. Rozkaz skoku warunkowego powoduje, że adres następnego rozkazu przewidzianego do pobrania jest nieznany. Wobec tego realizacja etapu pobierania może nastąpić dopiero po otrzymaniu adresu następnego rozkazu, który zostanie określony po zakończeniu etapu wykonywania. Potem na etapie wykonywania następuje oczekiwanie na pobranie kolejnego rozkazu.

Strata czasu z tego drugiego powodu może być zmniejszona przez zgadywanie. Prosta reguła jest następująca. Gdy rozkaz rozgałęzienia warunkowego przechodzi z etapu pobierania do etapu wykonywania, na etapie pobierania ma miejsce pobranie następnego rozkazu z pamięci po rozkazie rozgałęzienia. Wtedy, jeśli nie nastąpi rozgałęzienie, czas nie jest stracony. Gdy rozgałęzienie nastąpi, pobrany rozkaz musi być usunięty, a pobrany nowy.

Chociaż powyższe czynniki powodują zmniejszenia potencjalnej efektywności 2-etapowego potoku, pewne przyspieszenie jednak następuje. Aby uzyskać większe przyspieszenie, potok musi być przetwarzany na większej liczbie etapów. Rozważmy następującą dekompozycję przetwarzania rozkazu:

- **Pobranie rozkazu (FI).** Wczytanie następnego spodziewanego rozkazu do bufora.
- **Dekodowanie rozkazu (DI).** Określenie kodu operacji i specyfikatorów argumentu.
- **Obliczanie argumentów (CO).** Obliczenie efektywnego adresu każdego argumentu źródłowego. Może to obejmować obliczanie adresu z przesunięciem, adresu rejestrowego pośredniego, adresu pośredniego lub innych form.
- **Pobieranie argumentów (FO).** Pobranie każdego argumentu z pamięci. Argumenty w rejestrach nie muszą być pobierane.
- **Wykonanie rozkazu (EI).** Przeprowadzenie wskazanej operacji i przechowanie wyniku, jeśli taki jest, w ustalonej lokacji docelowej.
- **Zapisanie argumentu (WO).** Zapisanie wyniku w pamięci.

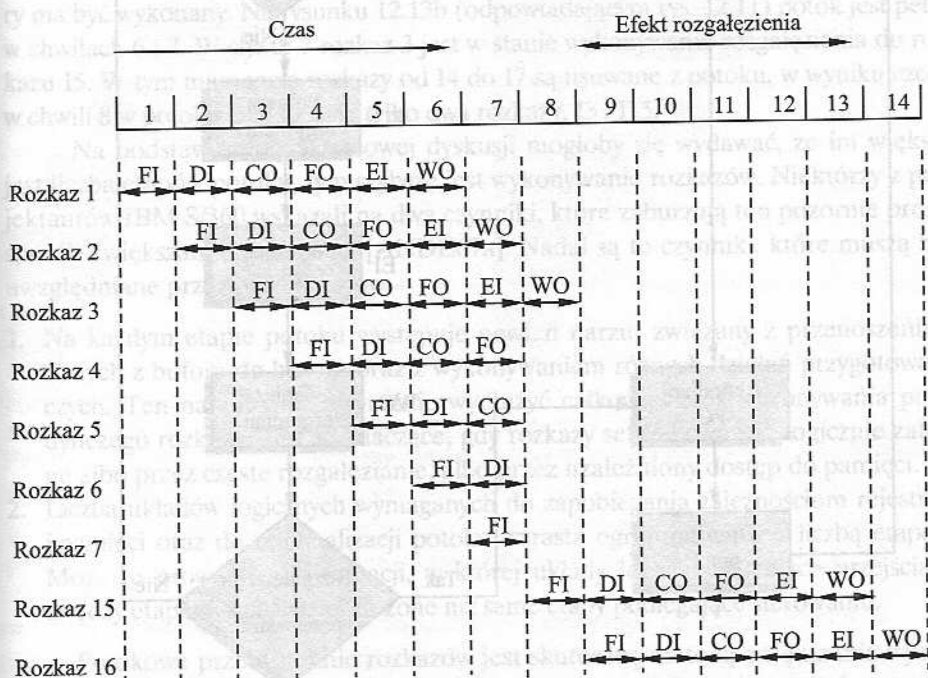


Rysunek 12.10. Przebieg czasowy przetwarzania potoku rozkazów

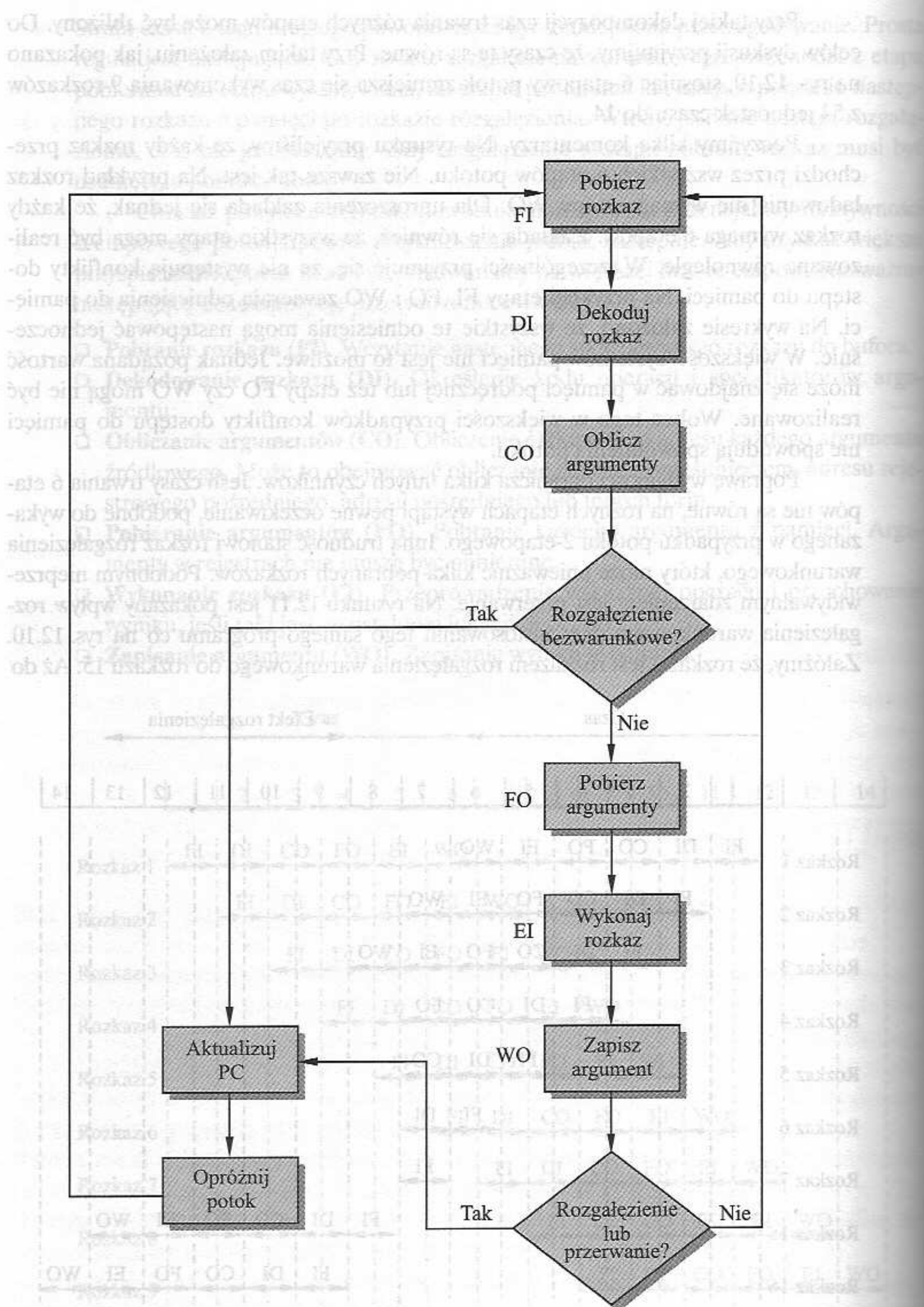
Przy takiej dekompozycji czas trwania różnych etapów może być zbliżony. Do celów dyskusji przyjmijmy, że czasy te są równe. Przy takim założeniu, jak pokazano na rys. 12.10, stosując 6-etapowy potok zmniejsza się czas wykonywania 9 rozkazów z 54 jednostek czasu do 14.

Poczyńmy kilka komentarzy. Na rysunku przyjęliśmy, że każdy rozkaz przechodzi przez wszystkich 6 etapów potoku. Nie zawsze tak jest. Na przykład rozkaz ładowania nie wymaga etapu WO. Dla uproszczenia zakłada się jednak, że każdy rozkaz wymaga 6 etapów. Zakłada się również, że wszystkie etapy mogą być realizowane równolegle. W szczególności przyjmuje się, że nie występują konflikty dostępu do pamięci. Na przykład etapy FI, FO i WO zawierają odniesienia do pamięci. Na wykresie założono, że wszystkie te odniesienia mogą następować jednocześnie. W większości systemów pamięci nie jest to możliwe. Jednak pożądana wartość może się znajdować w pamięci podręcznej lub też etapy FO czy WO mogą nie być realizowane. Wobec tego w większości przypadków konflikty dostępu do pamięci nie spowodują spowolnienia potoku.

Poprawę wydajności ogranicza kilka innych czynników. Jeśli czasy trwania 6 etapów nie są równe, na różnych etapach wystąpi pewne oczekiwanie, podobne do wykazanego w przypadku potoku 2-etapowego. Inną trudność stanowi rozkaz rozgałęzienia warunkowego, który może unieważnić kilka pobranych rozkazów. Podobnym nieprzewidywalnym zdarzeniem jest przerwanie. Na rysunku 12.11 jest pokazany wpływ rozgałęzienia warunkowego przy zastosowaniu tego samego programu co na rys. 12.10. Załóżmy, że rozkaz 3 jest rozkazem rozgałęzienia warunkowego do rozkazu 15. Aż do



Rysunek 12.11. Wpływ rozgałęzienia warunkowego na funkcjonowanie potoku rozkazów



Rysunek 12.12. **Sześcioetapowy potok rozkazów procesora**

wykonania tego rozkazu nie ma możliwości przewidzenia, który rozkaz będzie wykonywany jako następny. W tym przykładzie w potoku jest po prostu ładowany następny w kolejności rozkaz (rozkaz 4) i przetwarzanie jest kontynuowane. Na rysunku 12.10 rozgałęzienie nie następuje i uzyskujemy pełną poprawę wydajności. Na rysunku 12.11 rozgałęzienie następuje, jednak nie jest to wiadome aż do końca 7. jednostki czasu. W tej chwili potok musi być oczyszczony z bezużytecznych rozkazów. Podczas 8. jednostki czasu, rozkaz 15 wchodzi do potoku. W czasie od jednostki 9. do 12. nie kończy się wykonywanie żadnego rozkazu; jest to strata wydajności wynikająca z tego, że nie mogliśmy przewidzieć rozgałęzienia. Na rysunku 12.12 jest pokazany schemat logiczny przetwarzania potokowego uwzględniający rozgałęzienia i przerwania.

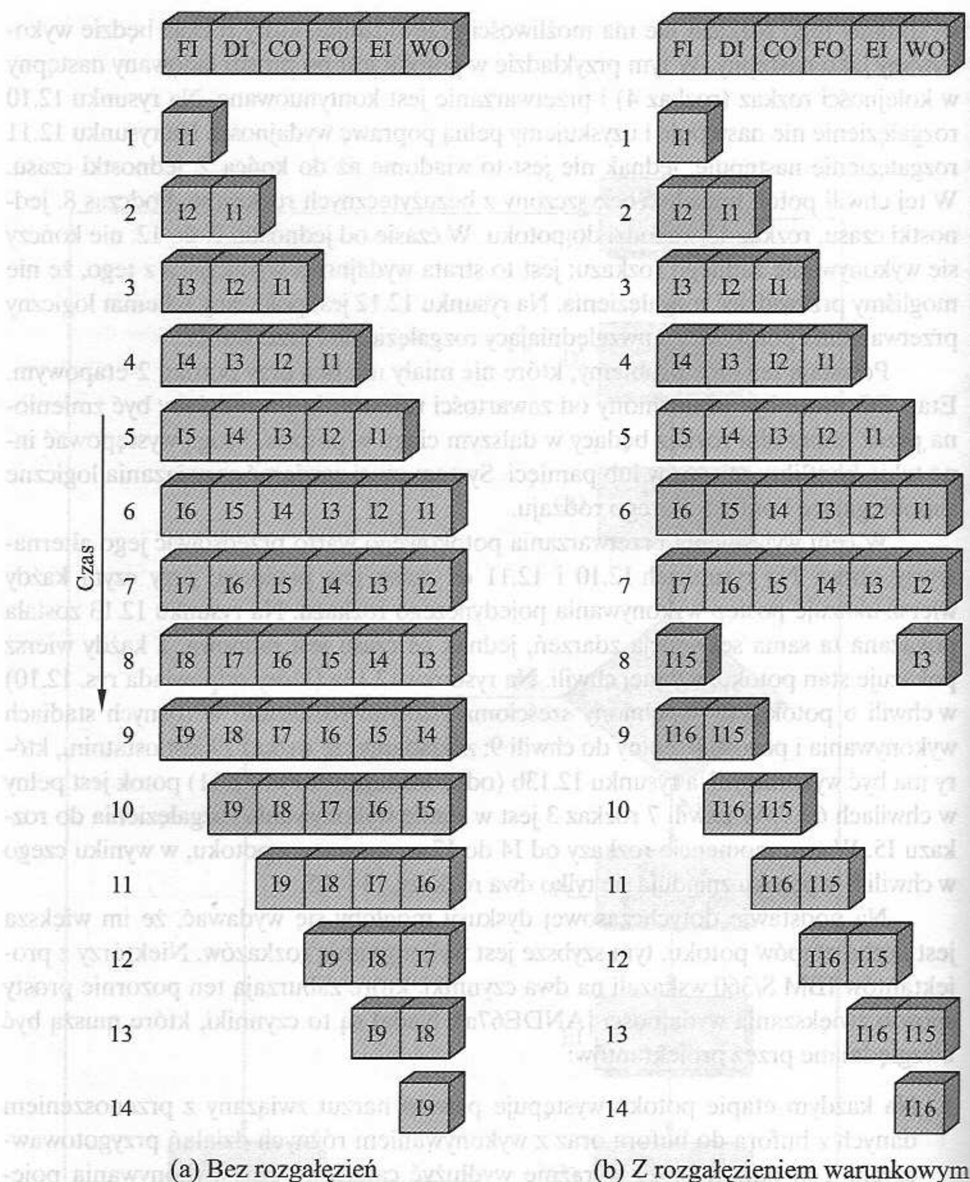
Powstają też inne problemy, które nie miały miejsca przy potoku 2-etapowym. Etap CO może być uzależniony od zawartości rejestru, która mogłaby być zmieniona przez poprzedni rozkaz będący w dalszym ciągu w potoku. Mogą występować inne takie konflikty rejestrów lub pamięci. System musi zawierać rozwiązania logiczne zapobiegające konfliktom tego rodzaju.

W celu wyjaśnienia przetwarzania potokowego warto przedstawić jego alternatywny obraz. Na rysunkach 12.10 i 12.11 oś czasu jest pozioma, przy czym każdy wiersz ukazuje postęp wykonywania pojedynczego rozkazu. Na rysunku 12.13 została pokazana ta sama sekwencja zdarzeń, jednak oś czasu jest pionowa, a każdy wiersz pokazuje stan potoku w danej chwili. Na rysunku 12.13a (który odpowiada rys. 12.10) w chwili 6 potok jest wypełniony sześcioma różnymi rozkazami w różnych stadiach wykonywania i pozostaje pełny do chwili 9; zakładamy, że rozkaz I9 jest ostatnim, który ma być wykonany. Na rysunku 12.13b (odpowiadającym rys. 12.11) potok jest pełny w chwilach 6 i 7. W chwili 7 rozkaz 3 jest w stanie wykonywania rozgałęzienia do rozkazu I5. W tym momencie rozkazy od I4 do I7 są usuwane z potoku, w wyniku czego w chwili 8 w potoku znajdują się tylko dwa rozkazy, I3 i I15.

Na podstawie dotychczasowej dyskusji mogłoby się wydawać, że im większa jest liczba etapów potoku, tym szybsze jest wykonywanie rozkazów. Niektórzy z projektantów IBM S/360 wskazali na dwa czynniki, które zaburzają ten pozornie prosty sposób zwiększania wydajności [ANDE67a]. Nadal są to czynniki, które muszą być uwzględniane przez projektantów:

1. Na każdym etapie potoku występuje pewien narzut związany z przenoszeniem danych z bufora do bufora oraz z wykonywaniem różnych działań przygotowawczych. Ten narzut może wyraźnie wydłużyć całkowity czas wykonywania pojedynczego rozkazu. Jest to znaczące, gdy rozkazy sekwencyjne są logicznie zależne albo przez częste rozgałęzianie, albo przez uzależniony dostęp do pamięci.
2. Liczba układów logicznych wymaganych do zapobiegania zależnościom rejestrów i pamięci oraz do optymalizacji potoku wzrasta ogromnie wraz z liczbą etapów. Może to prowadzić do sytuacji, w której układy logiczne sterujące przejściami między etapami są bardziej złożone niż same etapy podlegające sterowaniu.

Potokowe przetwarzanie rozkazów jest skuteczną metodą zwiększania wydajności, jednak wymaga starannego projektowania, aby osiągnąć optymalne wyniki przy rozsądnej złożoności.



Rysunek 12.13. Alternatywny obraz przetwarzania potokowego

Wydajność przetwarzania potokowego

W tym podrozdziale wprowadzimy pewne proste miary wydajności potoku i względnego przyspieszenia wynikającego z jego zastosowania (na podstawie analizy zawartej w [HWAN93]). Czas trwania cyklu τ potoku rozkazów jest czasem wymaganym do tego, aby dany zbiór rozkazów posunął się w potoku o jeden etap; każdy

kolumna na rys. 12.10 i 12.11 reprezentuje więc jeden cykl. Czas trwania cyklu można określić jako

$$\tau = \max_i[\tau_i] + d = \tau_m + d \qquad 1 \leqslant i \leqslant k$$

gdzie:

τ_m = maksymalne opóźnienie etapu (opóźnienie w tym etapie, w którym jest ono największe);

k = liczba etapów w potoku rozkazów;

d = opóźnienie zatrzasku konieczne do tego, aby sygnały i dane przeszły z jednego etapu do drugiego.

Ogólnie rzecz biorąc, opóźnienie d jest równoważne czasowi trwania impulsu zegara, a $\tau_m \gg d$. Załóżmy teraz, że przetwarzanych jest n rozkazów bez rozgałęzień. Łączny czas T_k wymagany do wykonania wszystkich n rozkazów wynosi

$$T_k = [k + (n-1)]\tau \qquad (12.1)$$

Do zakończenia wykonywania pierwszego rozkazu jest wymagana liczba cykli k, pozostałe zaś $n - 1$ rozkazów wymaga $n - 1$ cykli[2]. Równanie to można łatwo zweryfikować na podstawie rysunku 12.10. Dziewiąty rozkaz zostanie ukończony w 14 cyklu:

$$14 = [6 + (9 - 1)]$$

Współczynnik przyspieszenia potoku rozkazów w porównaniu z ich wykonywaniem bez przetwarzania potokowego jest definiowany jako

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{[k + (n-1)]\tau} = \frac{nk}{k + (n-1)} \qquad (12.2)$$

Na rysunku 12.14a współczynnik przyspieszenia został wykreślony w funkcji liczby rozkazów, które są wykonywane bez rozgałęzienia. Jak można było oczekiwać, przyspieszenie graniczne ($n \rightarrow \infty$) jest k-krotne. Na rysunku 12.14b pokazano współczynnik przyspieszenia jako funkcję liczby etapów w potoku rozkazów[3]. W tym przypadku współczynnik przyspieszenia sięga liczby rozkazów, jakie mogą być wprowadzone do potoku bez rozgałęzienia. Im większa jest zatem liczba etapów potoku, tym większe potencjalne przyspieszenie. W praktyce potencjalne korzyści z przetwarzania potokowego są ograniczane przez wzrost kosztów, opóźnienia między etapami oraz przez to, że napotkane rozgałęzienia wymagają opróżnienia potoku.

---

[2] Nie jesteśmy tutaj zbyt dokładni. Gdy wszystkie etapy będą wypełnione, czas cyklu będzie równy tylko maksymalnej wartości τ. Na początku czas cyklu związany z pierwszym lub z kilkoma pierwszymi cyklami może być mniejszy.

[3] Zwróćmy uwagę, że na rys. 12.14a oś x jest logarytmiczna, zaś na rys. 12.14b – liniowa.

Tabela 12.3. Zmiennopozycyjny rejestr stanu i sterowania procesora PowerPC

Bit	Definicja
0	Podsumowanie wyjątku. Ustawiany, gdy następuje jakikolwiek wyjątek; pozostaje ustawiony, aż do wyzerowania przez program.
1	Podsumowanie dozwolonego wyjątku. Ustawiany, gdy następuje jakikolwiek dozwolony wyjątek.
2	Podsumowanie wyjątku nieważnej operacji. Ustawiany, jeśli nastąpił wyjątek spowodowany przez nieważną operację.
3	Wyjątek przepełnienia. Wielkość wyniku przekracza tę, która może być reprezentowana.
4	Wyjątek niedomiaru. Wynik jest zbyt mały, aby mógł być znormalizowany.
5	Wyjątek dzielenia przez zero. Dzielnik jest zerem, a dzielna jest skończona i niezerowa.
6	Wyjątek niedokładności. Zaokrąglony wynik różni się od wyniku pośredniego lub następuje przepełnienie przy zablokowanym wyjątku przepełnienia.
7÷12	Wyjątek nieważnej operacji: 7: sygnalizująca NaN; 8: $(\infty - \infty)$); 9: $(\infty \div \infty)$; 10: $(0 \div 0)$; 11: $(\infty \times 0)$; 12: porównanie obejmujące NaN.
13	Zaokrąglenie ułamka. Zaokrąglenie wyniku inkrementowało ułamek.
14	Niedokładność ułamka. Zaokrąglenie wyniku zmieniło ułamek lub wystąpiło przepełnienie przy zablokowanym wyjątku przepełnienia.
15÷19	Znaczniki stanu wyniku. 5-bitowy kod określa: mniejszy niż, większy niż, równy, nieuporządkowany, cicha NaN, $\pm\infty$, $\pm$ znormalizowany, $\pm$ zdenormalizowany, ± 0.
20	Zarezerwowany.
21÷23	Wyjątek nieważnej operacji. 21 – zapotrzebowanie programowe; 22 – pierwiastek kwadratowy z liczby ujemnej; 23 – konwersja liczby całkowitej obejmująca wielką liczbę, nieskończoność lub NaN.
24	Zezwolenie wyjątku nieważnej operacji.
25	Zezwolenie wyjątku przepełnienia.
26	Zezwolenie wyjątku niedomiaru.
27	Zezwolenie wyjątku dzielenia przez zero.
28	Zezwolenie wyjątku niedokładności.
29	Tryb niezgodny z IEEE.
30÷31	Sterowanie zaokrąglaniem. 2-bitowy kod określa: do najbliższej, w stronę zera, w stronę $+\infty$, w stronę $-\infty$.

Niezacienione – bity stanu; zacienione – bity sterowania.

Jednostka przetwarzania rozgałęzień zawiera następujące rejestry widzialne dla użytkownika:

- **Rejestr warunku.** Zawiera osiem 4-bitowych pól kodu warunkowego (rys. 12.24b).
- **Rejestr powiązania.** Rejestr powiązania może być używany w operacjach rozgałęzienia warunkowego do adresowania pośredniego adresu docelowego. Może być również używany do operacji wywołania i powrotu. Jeżeli jest ustawiony bit LK w rozkazie rozgałęzienia warunkowego, to adres następujący po rozkazie rozgałęzienia jest umieszczany w rejestrze powiązania i może być użyty przy późniejszym powrocie.

czenia wykonywania tego rozkazu nie jest możliwe stwierdzenie, czy rozgałęzienie nastąpi, czy nie.

Zastosowano wiele sposobów postępowania z rozkazami rozgałęzienia warunkowego:

- zwielokrotnienie strumienia;
- pobieranie docelowego rozkazu z wyprzedzeniem;
- bufor pętli;
- przewidywanie rozgałęzienia;
- opóźnione rozgałęzienie.

Zwielokrotnione strumienie

W przypadku prostego potoku występują straty spowodowane przez rozkaz rozgałęzienia, ponieważ konieczne jest wybranie jednego z dwóch możliwych rozkazów i wybór ten może się okazać niewłaściwy. Brutalnym rozwiązaniem jest powielenie początkowych części potoku i umożliwienie równoczesnego pobrania obu rozkazów za pomocą dwóch strumieni. Rozwiązanie to stwarza pewne problemy:

- ❑ W przypadku zwielokrotnionego strumienia występują opóźnienia wynikające z rywalizacji o dostęp do rejestrów i pamięci.
- ❑ Następne rozkazy rozgałęzienia mogą wejść do potoku (którymkolwiek strumieniem), zanim zostanie przesądzone oryginalne rozgałęzienie. Każdy taki rozkaz wymaga dodatkowego strumienia.

Mimo powyższych wad strategia ta może poprawić wydajność. Przykładami maszyn z dwoma lub z większą ilością strumieni są IBM 370/168 oraz IBM 3033.

Pobieranie docelowego rozkazu z wyprzedzeniem

Gdy rozpoznawany jest rozkaz rozgałęzienia warunkowego, następuje wyprzedzające pobranie rozkazu docelowego razem z rozkazem następującym po rozgałęzieniu. Rozkaz docelowy jest następnie zachowywany aż do czasu wykonania rozkazu rozgałęzienia. W momencie rozgałęzienia rozkaz docelowy jest już pobrany.

Rozwiązanie to zastosowano w IBM 360/91.

Bufor pętli

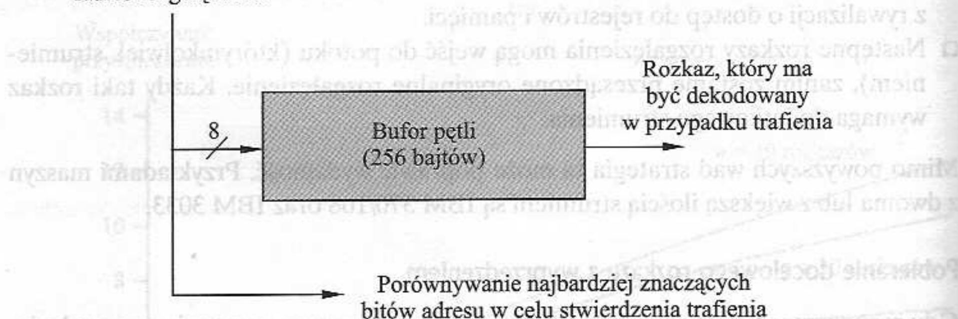
Bufor pętli jest małą, bardzo szybką pamięcią związaną z etapem pobierania rozkazów potoku. Zawiera on kolejno n ostatnio pobranych rozkazów. Jeśli ma nastąpić rozgałęzienie, sprawdza się najpierw, czy cel rozgałęzienia znajduje się wewnątrz bufora. Jeśli tak, to następny rozkaz jest pobierany z bufora. Stosowanie bufora pętli ma trzy zalety:

1. Dzięki pobieraniu z wyprzedzeniem bufor pętli zawiera pewne rozkazy następujące po kolei po adresie pobrania bieżącego rozkazu. Wobec tego kolejno pobierane rozkazy będą dostępne bez straty czasu wymaganego na dostęp do pamięci.

2. Jeśli następuje rozgałęzienie do rozkazu znajdującego się o kilka pozycji dalej od adresu rozkazu rozgałęzienia, to rozkaz docelowy będzie się już znajdował w buforze. Jest to użyteczne w przypadku stosunkowo powszechnych sekwencji IF-THEN (jeśli-to) oraz IF-THEN-ELSE (jeśli-to-w przeciwnym razie).
3. Taki sposób postępowania jest szczególnie przydatny przy operowaniu pętlami, stąd nazwa „bufor pętli". Jeżeli bufor ten jest dostatecznie pojemny, aby zmieścić wszystkie rozkazy składające się na pętlę, to rozkazy te muszą być pobrane tylko raz, przy pierwszej iteracji. W przypadku następnej iteracji wszystkie potrzebne rozkazy będą się już znajdowały w buforze.

Bufor pętli jest w zasadzie podobny do dedykowanej rozkazom pamięci podręcznej. Różnica polega na tym, że bufor pętli zawiera tylko sekwencje rozkazów i jest o wiele mniejszy, a wobec tego tańszy.

Na rysunku 12.15 jest pokazany przykład bufora pętli. Zawiera on 256 adresowalnych bajtów, zatem do jego indeksowania używa się 8 najmniej znaczących bitów. Pozostałe najbardziej znaczące bity są sprawdzane w celu stwierdzenia, czy cel rozgałęzienia leży wewnątrz wycinka przechwyconego przez bufor.



Rysunek 12.15. Bufor pętli

Wśród maszyn stosujących bufor pętli są niektóre maszyny CDC (Star 100, 6600, 7600) oraz CRAY-1. Wyspecjalizowana postać bufora pętli jest zaimplementowana w procesorze 68010 firmy Motorola i służy ona do wykonywania 3-rozkazowej pętli obejmującej rozkaz DBcc (*decrement and branch on condition* – dekrementacja i rozgałęzienie warunkowe – zobacz problem 12.6). Bufor ma pojemność 3 słów, a procesor powtarza wykonywanie tych rozkazów aż do spełnienia warunku pętli.

Przewidywanie rozgałęzień

Zastosowano wiele metod przewidywania, czy nastąpi rozgałęzienie. Wśród nich często używane są następujące:

- przewidywanie nigdy nie następującego rozgałęzienia;
- przewidywanie zawsze następującego rozgałęzienia;

- przewidywanie za pomocą kodu operacji;
- przełącznik nastąpiło/nie nastąpiło;
- tablica historii rozgałęzień.

Pierwsze trzy rozwiązania są statyczne: nie zależą one od historii poprzedzającej rozkaz rozgałęzienia warunkowego. Ostatnie dwa są dynamiczne: zależą od historii.

Pierwsze dwa rozwiązania są najprostsze. Albo zakłada się zawsze, że rozgałęzienie nie nastąpi i kontynuuje się kolejne pobieranie rozkazów. Albo zakłada się zawsze, że rozgałęzienie nastąpi i pobiera się rozkaz w celu rozgałęzienia. Maszyny 68020 i VAX 11/780 stosują przewidywanie nigdy nie następującego rozgałęzienia. VAX 11/780 umożliwia też minimalizowanie efektu złej decyzji. Jeśli pobranie rozkazu po rozgałęzieniu spowoduje błąd strony lub naruszenie ochrony, procesor wstrzymuje wstępne pobieranie aż do upewnienia się, że rozkaz ten powinien być pobrany.

Badania nad zachowaniem się programów wykazały, że rozgałęzienia warunkowe zabierają ponad 50% czasu [LILJ88], a wobec tego systematyczne pobieranie wstępne docelowego rozkazu rozgałęzienia powinno zapewnić większą wydajność niż systematyczne pobieranie wstępne z kolejnych, następujących po sobie rozkazów. Jednak w przypadku maszyn realizujących stronicowanie pobieranie wstępne docelowego rozkazu rozgałęzienia powoduje większe prawdopodobieństwo błędu strony niż kolejne pobieranie wstępne następnego rozkazu. Powinna więc być wzięta pod uwagę wynikająca stąd strata wydajności. Może być zastosowany mechanizm zapobiegawczy umożliwiający zmniejszenie tej straty.

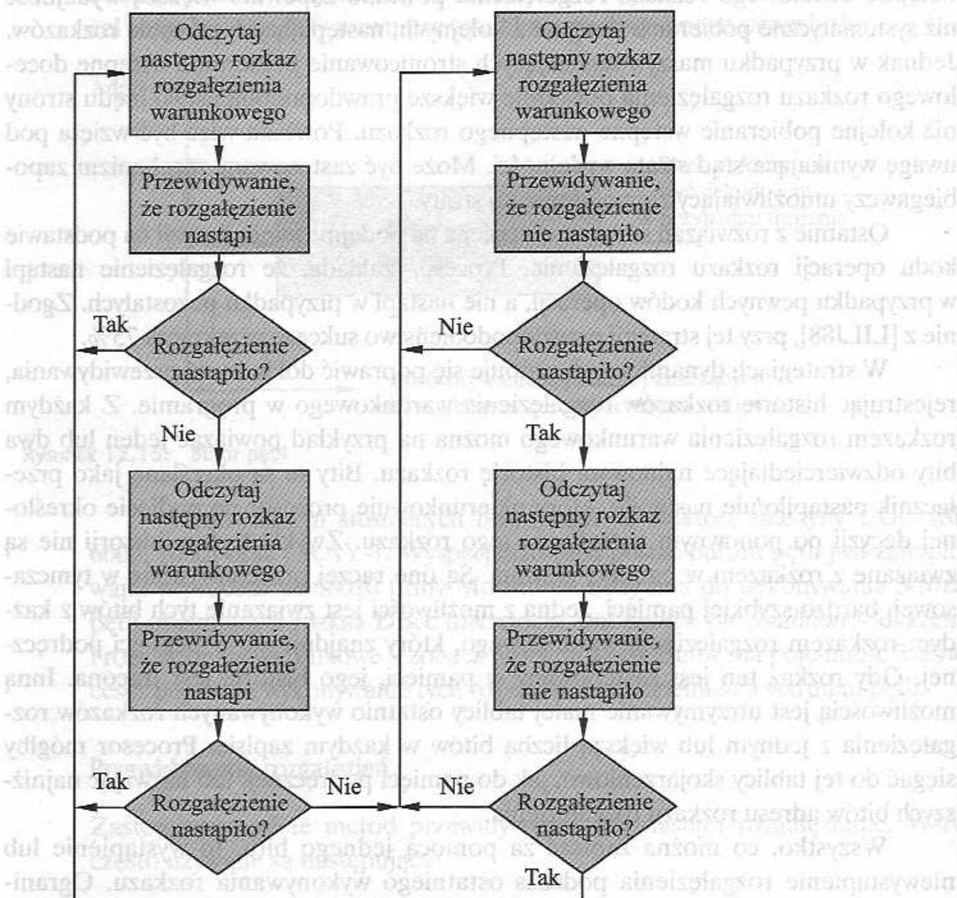
Ostatnie z rozwiązań statycznych polega na podejmowaniu decyzji na podstawie kodu operacji rozkazu rozgałęzienia. Procesor zakłada, że rozgałęzienie nastąpi w przypadku pewnych kodów operacji, a nie nastąpi w przypadku pozostałych. Zgodnie z [LILJ88], przy tej strategii prawdopodobieństwo sukcesu przekracza 75%.

W strategiach dynamicznych próbuje się poprawić dokładność przewidywania, rejestrując historię rozkazów rozgałęzienia warunkowego w programie. Z każdym rozkazem rozgałęzienia warunkowego można na przykład powiązać jeden lub dwa bity odzwierciedlające najnowszą historię rozkazu. Bity te są określane jako przełącznik nastąpiło/nie nastąpiło, który ukierunkowuje procesor na podjęcie określonej decyzji po ponownym natrafieniu tego rozkazu. Zwykle te bity historii nie są związane z rozkazem w pamięci głównej. Są one raczej przechowywane w tymczasowej, bardzo szybkiej pamięci. Jedną z możliwości jest związanie tych bitów z każdym rozkazem rozgałęzienia warunkowego, który znajduje się w pamięci podręcznej. Gdy rozkaz ten jest zastępowany w pamięci, jego historia jest tracona. Inną możliwością jest utrzymywanie małej tablicy ostatnio wykonywanych rozkazów rozgałęzienia z jednym lub większą liczbą bitów w każdym zapisie. Procesor mógłby sięgać do tej tablicy skojarzeniowo, jak do pamięci podręcznej, lub używając najniższych bitów adresu rozkazu rozgałęzienia.

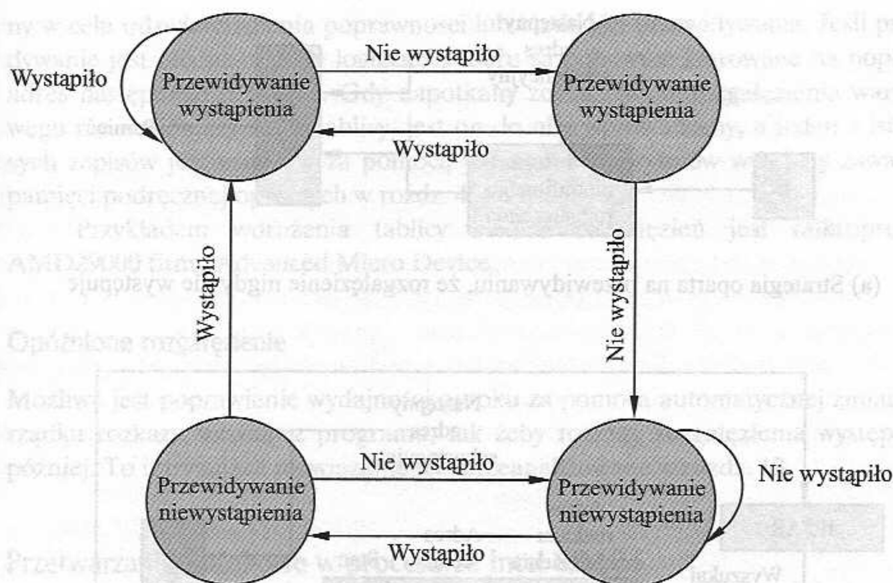
Wszystko, co można zapisać za pomocą jednego bitu, to wystąpienie lub niewystąpienie rozgałęzienia podczas ostatniego wykonywania rozkazu. Ograniczenie wynikające ze stosowania pojedynczego bitu ujawnia się w razie rozkazu rozgałęzienia warunkowego, które następuje prawie zawsze, na przykład w przy-

padku rozkazu pętli. Przy zastosowaniu jednego bitu historii, błędne przewidywanie wystąpi dwukrotnie przy każdym użyciu pętli: raz przy wejściu do pętli i raz przy jej opuszczeniu.

Jeśli stosowane są dwa bity, mogą one być użyte do zarejestrowania wyników ostatnich dwóch przypadków wykonania stowarzyszonego rozkazu lub do zapisania stanu w pewien inny sposób. Na rysunku 12.16 jest pokazane typowe rozwiązanie (inne możliwości są opisane w problemie 12.5). Załóżmy, że algorytm rozpoczyna się od górnego lewego rogu wykresu. Za każdym razem, gdy zostanie napotkany rozkaz rozgałęzienia warunkowego, w procesie decyzyjnym przewiduje się, że następnym rozkazem będzie rozgałęzienie. Jeśli jedno takie przewidywanie okaże się błędne, zgodnie z tym algorytmem zakłada się nadal, że kolejnym rozkazem będzie rozgałęzienie. Dopiero gdy dwa razy pod rząd nie nastąpiło rozgałęzienie, w algorytmie przechodzi się do prawej strony wykresu. Następnie algorytm ten będzie przewidywał, że rozgałęzienia nie nastąpią, aż wystąpią one dwa razy pod rząd. Algorytm ten wymaga więc dwóch kolejnych niewłaściwych przewidywań, aby zaszła zmiana przewidywania.



Rysunek 12.16. Sieć działań przewidywania rozgałęzień



Rysunek 12.17. Graf stanów przewidywania rozgałęzień

Proces decyzyjny może być reprezentowany za pomocą maszyny o skończonej liczbie stanów, pokazanej na rys. 12.17. Tego rodzaju reprezentacja jest powszechnie stosowana w literaturze.

Opisane właśnie stosowanie bitów historii ma jedną wadę: jeśli zostaje podjęta decyzja, że nastąpi rozgałęzienie, rozkaz docelowy nie może być pobrany, zanim jego adres, który jest argumentem w rozkazie rozgałęzienia warunkowego, nie zostanie zdekodowany. Można by osiągnąć większą efektywność, jeśli pobieranie rozkazu mogłoby być zapoczątkowane tuż po podjęciu decyzji rozgałęzienia. Do tego celu trzeba zachowywać więcej informacji w miejscu nazywanym *buforem docelowym rozgałęzienia* lub *tablicą historii rozgałęzień*.

Tablica historii rozgałęzień jest małą pamięcią podręczną powiązaną z etapem pobierania rozkazu w potoku. Każdy zapis w tablicy składa się z trzech elementów: adresu rozkazu rozgałęzienia, pewnej liczby bitów historii, wskazujących na używanie tego rozkazu, oraz informacji o rozkazie docelowym. W większości propozycji i wdrożeń to trzecie pole zawiera adres rozkazu docelowego. Inną możliwością jest to, żeby trzecie pole zawierało sam rozkaz docelowy. Wymienność jest oczywista: przechowywanie adresu prowadzi do mniejszej tablicy, jednak równocześnie do wydłużenia czasu pobierania rozkazu w porównaniu z przechowywaniem rozkazu docelowego [RECH98].

Na rysunku 12.18 widać odmienność tego rozwiązania w porównaniu z przewidywaniem nigdy nie następującego rozgałęzienia. W ramach tej ostatniej strategii pobierany jest zawsze następny adres w kolejności. Jeśli następuje rozgałęzienie, to wykrywają to pewne układy logiczne procesora i przekazują instrukcję, że następny rozkaz powinien być pobrany spod adresu docelowego (obok polecenia oczyszczenia

(a) Strategia oparta na przewidywaniu, że rozgałęzienie nigdy nie występuje

(b) Strategia oparta na tablicy historii rozgałęzień

Rysunek 12.18. Postępowanie z rozgałęzieniami

potoku). Tablica historii rozgałęzień jest traktowana jak pamięć podręczna. Każde wstępne pobranie powoduje zajrzenie do tablicy historii rozgałęzień. Jeśli nie znaleziono zgodności, do pobierania jest używany następny adres w kolejności. Jeśli stwierdzono zgodność, przewidywanie jest oparte na stanie rozkazu: do układów logicznych wyboru jest kierowany albo następny adres w kolejności, albo docelowy adres rozgałęzienia.

Gdy rozkaz rozgałęzienia zostanie wykonany, z etapu wykonywania jest kierowany sygnał o wyniku do tablicy historii rozgałęzień. Stan rozkazu jest aktualizowa-

ny w celu odzwierciedlenia poprawności lub błędności przewidywania. Jeśli przewidywanie jest błędne, układy logiczne wyboru są ponownie kierowane na poprawny adres następnego pobrania. Gdy napotkany został rozkaz rozgałęzienia warunkowego nie znajdujący się w tablicy, jest on do niej wprowadzany, a jeden z istniejących zapisów jest usuwany za pomocą jednego z algorytmów wymiany zawartości pamięci podręcznej opisanych w rozdz. 4.

Przykładem wdrożenia tablicy historii rozgałęzień jest mikroprocesor AMD29000 firmy Advanced Micro Device.

Opóźnione rozgałęzienie

Możliwe jest poprawienie wydajności potoku za pomocą automatycznej zmiany porządku rozkazu wewnątrz programu, tak żeby rozkazy rozgałęzienia występowały później. To intrygujące rozwiązanie jest przeanalizowane w rozdz. 13.

Przetwarzanie potokowe w procesorze Intel 80486

W procesorze 80486 wdrożono 5-etapowy potok o następujących etapach:

- **Pobieranie.** Rozkazy są pobierane z pamięci podręcznej lub zewnętrznej i umieszczane w jednym z dwóch 16-bajtowych buforów wstępnego pobierania. Celem etapu pobierania jest napełnienie tych buforów nowymi danymi natychmiast po wykorzystaniu starych danych przez dekoder rozkazu. Ponieważ rozkazy mają zmienną długość (od 1 do 11 bajtów, nie licząc przedrostków), stan etapu pobierania w stosunku do innych etapów potoku zmienia się od rozkazu do rozkazu. Przeciętnie jest pobieranych około 5 rozkazów przy ładowaniu każdych 16 bajtów [CRAW90]. Etap pobierania przebiega niezależnie od innych etapów, co ma na celu utrzymywanie pełnych buforów wstępnego pobierania.
- **Etap dekodowania 1.** Wszystkie kody operacji oraz informacje o trybie adresowania są dekodowane podczas etapu D1. Wymagane informacje, również dotyczące długości rozkazu, są zawarte w co najwyżej 3 pierwszych bajtach rozkazu. Wobec tego właśnie 3 bajty są przenoszone z buforów wstępnego pobierania w czasie etapu D1. Dekoder D1 może następnie sprawić, że na etapie D2 nastąpi przechwycenie pozostałej części rozkazu (przesunięcia i danych natychmiastowych), która nie była angażowana w dekodowanie prowadzone podczas etapu D1.
- **Etap dekodowania 2.** Podczas etapu D2 jest poszerzany każdy kod operacji i są tworzone sygnały sterujące ALU. Zachodzi także sterowanie obliczaniem bardziej złożonych trybów adresowania.
- **Wykonywanie (execute, EX).** Etap ten obejmuje operacje ALU, dostęp do pamięci podręcznej i aktualizację rejestru.
- **Zapis opóźniony (write back, WB).** Na tym etapie, jeśli jest potrzebny, następuje aktualizacja rejestrów i znaczników stanu zmienionych na poprzednim etapie (wykonywania). Jeśli bieżący rozkaz aktualizuje pamięć, obliczona wartość jest kierowana jednocześnie do pamięci podręcznej i do buforów zapisu interfejsu magistrali.

Dzięki zastosowaniu dwóch etapów dekodowania potok może utrzymywać przepustowość zbliżoną do jednego rozkazu na cykl zegara. Złożone rozkazy oraz rozkazy rozgałęzienia warunkowego mogą zmniejszyć tę przepustowość.

Na rysunku 12.19 są pokazane przykłady działania potoku. W części (a) widać, że dostęp do pamięci nie powoduje opóźnienia potoku. Jak jednak wynika z części (b), może wystąpić opóźnienie w przypadku wartości używanych do obliczania adresu pamięci. To znaczy, jeśli wartość jest ładowana z pamięci do rejestru, a rejestr ten jest następnie używany jako rejestr podstawowy w następnym rozkazie, to procesor będzie czekał przez jeden cykl. W tym przykładzie procesor sięga do pamięci na etapie EX pierwszego rozkazu i przechowuje pobraną wartość w rejestrze podczas cyklu WB. Jednak następny rozkaz potrzebuje tego rejestru na swoim etapie D2. Gdy etap D2 zrównuje się z etapem WB poprzedniego rozkazu, obejściowa ścieżka sygnału pozwala na to, że na etapie D2 będzie miał miejsce dostęp do tych samych danych co używane podczas zapisu na etapie WB; pozwala to na zaoszczędzenie jednego etapu potoku.

Na rysunku 12.19c jest pokazane taktowanie rozkazu rozgałęzienia przy założeniu, że następuje rozgałęzienie. Rozkaz porównania aktualizuje kody warunkowe na etapie WB, zaś ścieżki obejściowe udostępniają je jednocześnie na etapie EX rozkazu skoku. Równolegle, podczas etapu EX rozkazu skoku, procesor realizuje

Pobieranie	D1	D2	EX	WB			MOV Reg1, Mem1
	Pobieranie	D1	D2	EX	WB		MOV Reg1, Reg2
		Pobieranie	D1	D2	EX	WB	MOV Mem2, Reg1

(a) Brak opóźnienia związanego z ładowaniem danych

Pobieranie	D1	D2	EX	WB		MOV Reg1, Mem1
	Pobieranie	D1		D2	EX	ADD Reg2, (Reg1)

(b) Opóźnienie związane z ładowaniem wskaźnika

Pobieranie	D1	D2	EX	WB				CMP Reg1 Imm
	Pobieranie	D1	D2	EX				Jcc Target
				Pobieranie	D1	D2	EX	Target

(c) Przebieg czasowy rozkazu rozgałęzienia

Rysunek 12.19. Przykłady potoku rozkazów procesora 80486

spekulatywny cykl pobierania adresu docelowego skoku. Jeśli procesor stwierdzi fałszywość warunku rozgałęzienia, porzuca wstępne pobieranie i kontynuuje pracę z następnym rozkazem sekwencyjnym (już pobranym i zdekodowanym).

12.5. Procesor Pentium

Ogólny widok organizacji procesora Pentium jest pokazany na rys. 4.13. W tym podrozdziale przeanalizujemy pewne rozwiązania szczegółowe.

Organizacja rejestrów

Organizacja rejestrów obejmuje następujące rodzaje rejestrów (tab. 12.1):

- **Ogólnego przeznaczenia (robocze).** Występuje osiem 32-bitowych rejestrów ogólnego przeznaczenia (patrz rys. 12.3c). Mogą one być używane przez wszystkie rodzaje rozkazów Pentium; mogą także służyć do przechowywania argumentów do obliczania adresu. Ponadto niektóre z tych rejestrów mogą służyć do celów specjalnych. Na przykład rozkazy łańcuchowe używają zawartości rejestrów ECX, ESI i EDI jako argumentów, bez konieczności jawnego odnoszenia się do tych rejestrów w rozkazie. Dzięki temu wiele rozkazów można zakodować zwięźlej.
- **Segmentowe.** Sześć 16-bitowych rejestrów segmentowych zawiera selektory segmentu służące jako indeksy w tablicach segmentów, co omówiliśmy w rozdz. 8. Rejestr segmentu kodu (CS) odnosi się do segmentu zawierającego wykonywany rozkaz. Rejestr segmentu stosu (SS) odnosi się do segmentu zawierającego stos widzialny dla użytkownika. Pozostałe rejestry segmentowe (DS, ES, FS, GS) umożliwiają użytkownikowi równoczesne odniesienie do czterech oddzielnych segmentów danych.
- **Znaczniki stanu.** Rejestr EFLAGS zawiera kody warunkowe i różne bity trybu.
- **Wskaźnik rozkazu.** Zawiera adres bieżącego rozkazu.

Są też rejestry specjalnie przeznaczone do współpracy z jednostką zmiennopozycyjną:

- **Numeryczne.** Każdy rejestr zawiera 80-bitową liczbę zmiennopozycyjną o zwiększonej dokładności. Istnieje 8 rejestrów, które działają jako stos, przy czym w liście rozkazów występują rozkazy umieszczania i zdejmowania.
- **Sterowania.** 16-bitowy rejestr sterowania zawiera bity sterujące pracą jednostki zmiennopozycyjnej, łącznie z zaokrąglaniem; bity określające dokładność pojedynczą, podwójną i rozszerzoną oraz bity zezwolenia i blokowania różnych warunków wyjątkowych.
- **Stanu.** 16-bitowy rejestr stanu zawiera bity odzwierciedlające stan bieżący jednostki zmiennopozycyjnej, włącznie z 3-bitowym wskaźnikiem wierzchołka stosu; kody warunkowe informujące o wyniku ostatniej operacji oraz znaczniki stanu wyjątku.

- **Słowo wyróżników.** Rejestr 16-bitowy zawiera 2-bitowy wyróżnik dla każdego numerycznego rejestru zmiennopozycyjnego, który wskazuje rodzaj zawartości odpowiedniego rejestru. Cztery możliwe wartości obejmują: ważny, zerowy, specjalny (NaN, nieskończoność, zdenormalizowany) i pusty. Wyróżniki te umożliwiają programom sprawdzanie zawartości rejestru numerycznego bez skomplikowanego dekodowania aktualnych danych w rejestrze.

Tabela 12.1. Rejestry procesora Pentium

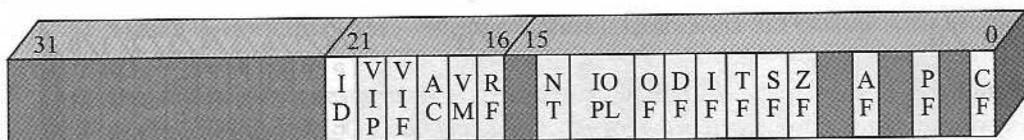
(a) Jednostka całkowitoliczbowa			
Rodzaj	**Liczba**	**Długość (bitów)**	**Przeznaczenie**
Robocze	8	32	Robocze rejestry użytkownika
Segmentowe	6	16	Zawierają selektory segmentu
Znaczniki stanu	1	32	Bity stanu i sterowania
Wskaźnik rozkazu	1	32	Wskaźnik rozkazu
(b) Jednostka zmiennopozycyjna			
Rodzaj	**Liczba**	**Długość (bitów)**	**Przeznaczenie**
Numeryczne	8	80	Przechowywanie liczb zmiennopozycyjnych
Sterowania	1	16	Bity sterowania
Stanu	1	16	Bity stanu
Słowo wyróżników	1	16	Określa zawartość rejestrów numerycznych
Wskaźnik rozkazu	1	48	Wskazuje rozkaz przerwany przez wyjątek
Wskaźnik danych	1	48	Wskazuje argument przerwany przez wyjątek

Zastosowanie większości z powyższych rejestrów jest łatwe do zrozumienia. Opiszemy krótko kilka z nich.

Rejestr EFLAGS

Rejestr EFLAGS (rys. 12.20) pokazuje stan procesora i pomaga sterować jego pracą. Obejmuje sześć kodów stanu zdefiniowanych w tabeli 10.8 (przeniesienie, parzystość, pomocniczy, zero, znak, przepełnienie), które informują o wynikach operacji na liczbach całkowitych. Ponadto występują tu bity, które można określić mianem kontrolnych; należą do nich:

- **Znacznik pułapki (TF** – *Trap Flag*). Gdy jest ustawiony, powoduje przerwanie po wykonaniu każdego rozkazu. Jest używany przy usuwaniu błędów z programów.
- **Znacznik zezwolenia przerwania (IF** – *Interrupt Enable Flag*). Gdy jest ustawiony, procesor rozpozna przerwania zewnętrzne.
- **Znacznik kierunku (DF** – *Direction Flag*). Wskazuje, czy rozkazy przetwarzania łańcuchowego inkrementują, czy dekrementują stan półrejestrów 16-bitowych SI i DI (w przypadku operacji 16-bitowych) lub rejestrów 32-bitowych ESI i EDI (w przypadku operacji 32-bitowych).



ID – znacznik stanu identyfikacji
VIP – zawieszenie wirtualnego przerwania
VIF – znacznik wirtualnego przerwania
AC – sprawdzenie wyrównania
VM – tryb wirtualnego 8086
RF – znacznik stanu
NT – znacznik stanu zadania zagnieżdżonego
IOPL – poziom uprzywilejowania wejścia-wyjścia
OF – znacznik stanu przepełnienia
DF – znacznik stanu kierunku
IF – znacznik stanu zezwolenia przerwania
TF – znacznik stanu pułapki
SF – znacznik znaku
ZF – znacznik stanu zera
AF – pomocniczy znacznik stanu przeniesienia
PF – znacznik stanu parzystości
CF – znacznik stanu przeniesienia

Rysunek 12.20. Rejestr EFLAGS procesora Pentium II

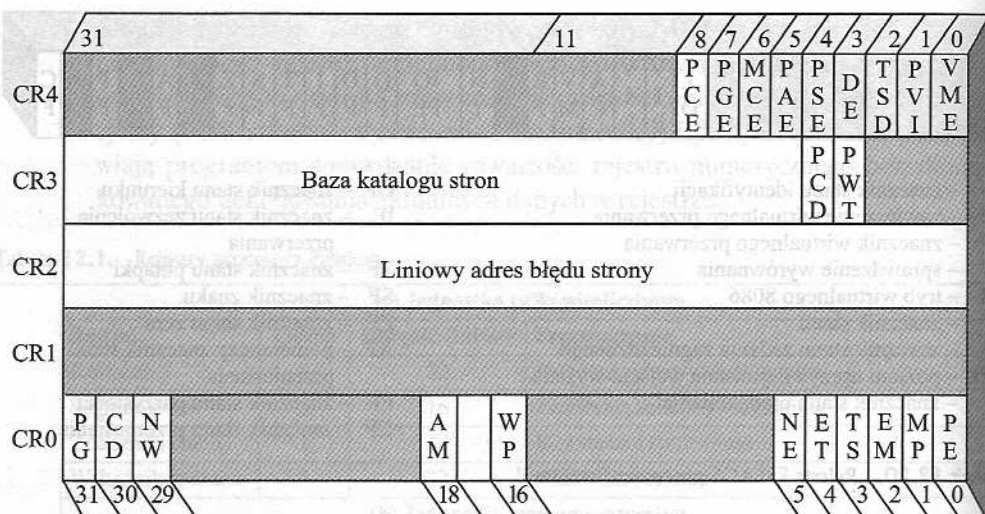
- **Znacznik uprzywilejowania wejścia-wyjścia (IOPL** – *I/O Privilege Flag*). Gdy jest ustawiony, powoduje, że procesor generuje sygnał wyjątku w stosunku do wszystkich dostępów do urządzeń wejścia-wyjścia podczas funkcjonowania w trybie chronionym.
- **Znacznik wznowienia** (**RF** – *Resume Flag*). Umożliwia programiście zablokowanie wyjątków związanych z usuwaniem błędów, co pozwala na wznowienie rozkazu bez natychmiastowego powodowania następnego wyjątku.
- **Kontrola wyrównania** (**AC** – *Alignment Check*). Jest wzbudzany, jeśli słowo lub podwójne słowo jest adresowane w nieodpowiednich granicach.
- **Znacznik identyfikacji** (**ID** – *Idenfication Flag*). Bit ten może być ustawiony i wyzerowany, jeśli procesor może wykonywać rozkaz CPUID. Rozkaz ten dostarcza informacji o sprzedawcy, rodzinie maszyn i o modelu.

Ponadto występują 4 bity związane z trybem operacji. Znacznik stanu zadania zagnieżdżonego (NT) wskazuje, że bieżące zadanie jest zagnieżdżone wewnątrz innego zadania w trybie chronionym. Bit trybu wirtualnego (VM) umożliwia programiście zezwolenie lub zablokowanie trybu wirtualnego 8086, w którym procesor pracuje jak 8086. Znacznik stanu przerwania wirtualnego (VIF) oraz znacznik stanu zawieszenia przerwania wirtualnego (VIP) są używane w środowisku wielozadaniowym.

Rejestry sterowania

Pentium wykorzystuje cztery 32-bitowe rejestry sterowania (rejestr CR1 jest nieużywany) w celu sterowania różnymi aspektami pracy procesora (rys. 12.21). Rejestr CR0 zawiera znaczniki stanu sterowania systemowego, które sterują trybami lub wskazują stany odnoszące się raczej do procesora niż do wykonywania indywidualnego zadania. Znaczniki te są następujące:

- **Zezwolenie ochrony** (**PE** – *Protection Enable*). Zezwala lub blokuje tryb pracy chronionej.



PCE – zezwolenie licznika wydajności	PG – stronicowanie
PGE – zezwolenie stron globalnych	CD – blokada pamięci podręcznej
MCE – zezwolenie sprawdzenia komputera	NW – niestosowanie zapisu jednoczesnego
PAE – fizyczne poszerzenie adresu	AM – maska wyrównania
PSE – rozszerzenia rozmiaru strony	WP – ochrona zapisu
DE – rozszerzenia uruchamiania (debugging)	NE – błąd numeryczny
TSD – blokada znaku czasowego	ET – rodzaj rozszerzenia
PVI – przerwania wirtualne w trybie chronionym	TS – przełączenie zadania
VME – rozszerzenia trybu wirtualnego 8086	EM – emulacja
PCD – blokada pamięci podręcznej na poziomie strony	MP – koprocesor kontrolny
PWT – przejrzystość zapisów na poziomie strony	PE – zezwolenie ochrony

Rysunek 12.21. Rejestry sterowania procesora Pentium II

- **Monitorowanie koprocesora (MP** – *Monitor Coprocessor*). Istotny tylko przy realizacji programów z dawniejszych maszyn Pentium; odnosi się do obecności koprocesora arytmetycznego.
- **Emulacja (EM** – *Emulation*). Ustawiany, gdy procesor nie ma jednostki zmiennopozycyjnej. Powoduje przerwanie, gdy czyniona jest próba wykonania rozkazów zmiennopozycyjnych.
- **Przełączone zadanie (TS** – *Task Switched*). Wskazuje, że procesor przełączył zadania.
- **Rodzaj rozszerzenia (ET** – *Extension Type*). Nie używany w Pentium; wskazuje obsługę koprocesora arytmetycznego we wcześniejszych maszynach.
- **Błąd numeryczny (NE** – *Numeric Error*). Umożliwia standardowy mechanizm zgłaszania błędów zmiennopozycyjnych na zewnętrznych liniach danych.
- **Ochrona zapisu (WP** – *Write Protect*). Gdy ten bit jest zerem, strony tylko do odczytu na poziomie użytkownika mogą być zapisane przez program nadzorczy. Cecha ta jest użyteczna przy wspieraniu tworzenia procesów w pewnych systemach operacyjnych.

- **Maska wyrównania** (AM – *Alignment Mask*). Umożliwia lub blokuje sprawdzanie wyrównania.
- **Bez zapisu jednoczesnego** (NW – *Not Write through*). Wybiera tryb pracy pamięci podręcznej danych. Gdy ten bit jest ustawiony, operacje zapisu jednoczesnego pamięci podręcznej są zablokowane.
- **Blokowanie pamięci podręcznej** (CD – *Cache Disable*). Umożliwia lub blokuje mechanizm zapełniania wewnętrznej pamięci podręcznej.
- **Stronicowanie** (PG – *Paging*). Umożliwia lub blokuje stronicowanie.

Gdy stronicowanie jest dozwolone, rejestry CR2 i CR3 są ważne. Rejestr CR2 służy do przechowywania 32-bitowego adresu liniowego ostatniej strony, do której nastąpiło odniesienie przed przerwaniem spowodowanym przez błąd strony. W rejestrze CR3 w pierwszych 20 bitach liczonych od lewej strony przechowuje się 20 najbardziej znaczących bitów adresu podstawowego katalogu stron; pozostałość adresu stanowią zera. Dwa bity CR3 są używane do sterowania końcówek kontrolujących pracę zewnętrznej pamięci podręcznej. Blokowanie pamięci podręcznej na poziomie strony (PCD) umożliwia lub blokuje dostęp do zewnętrznej pamięci podręcznej, a bit przezroczystości zapisu na poziomie stron (PWT) steruje zapisem jednoczesnym w zewnętrznej pamięci podręcznej.

W rejestrze CR4 jest zdefiniowanych 9 dodatkowych bitów kontrolnych:

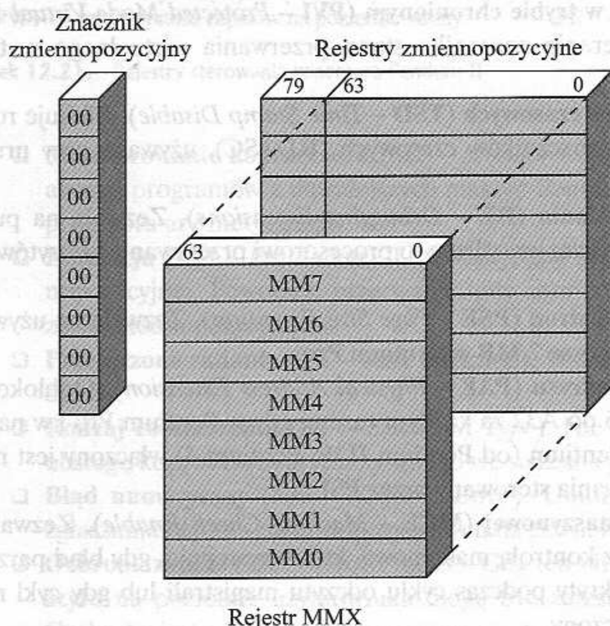
- **Rozszerzenie trybu wirtualnego 8086** (VME – *Virtual-8086 Mode Extension*). Zezwala wspieranie znacznika stanu przerwania wirtualnego w trybie wirtualnym 8086.
- **Przerwania wirtualne w trybie chronionym** (PVI – *Protected Mode Virtual Interrupts*). Zezwala wspieranie znacznika stanu przerwania wirtualnego w trybie chronionym.
- **Blokowanie znaczników czasowych** (TSD – *Time Stamp Disable*). Blokuje rozkaz odczytywania licznika znaczników czasowych (RDTSC), używany przy uruchamianiu programów.
- **Rozszerzenie uruchamiania** (DE – *Debugging Extensions*). Zezwala na punkty przerwania wejścia-wyjścia; umożliwia to procesorowi przerywanie odczytów i zapisów wejścia-wyjścia.
- **Rozszerzenie rozmiaru stron** (PSE – *Page Size Extension*). Zezwala na używanie stron 4 MB w Pentium oraz 2 MB w Pentium Pro.
- **Fizyczne poszerzenie adresu** (PAE – *Physical Address Extension*). Odblokowuje linie adresowe od A35 do A32 za każdym razem, gdy w Pentium Pro i w następnych architekturach Pentium (od Pentium II do Pentium 4) włączony jest nowy, specjalny tryb adresowania sterowany przez PSE.
- **Zezwolenie kontroli maszynowej** (MCE – *Machine Check Enable*). Zezwala na przerwanie związane z kontrolą maszynową, które występuje, gdy błąd parzystości danych zostaje wykryty podczas cyklu odczytu magistrali lub gdy cykl magistrali nie został zakończony.

- **Zezwolenie stron globalnych** (**PGE** – *Page Global Enable*). Zezwala na używanie stron globalnych. Gdy PGE=1 i realizowane jest przełączanie zadań, wszystkie wpisy TLB są usuwane, z wyjątkiem oznaczonych jako globalne.
- **Zezwolenie licznika wydajności** (**PCE** – *Preformance Counter Enable*). Zezwala na wykonywanie rozkazu RDPMC (odczytaj licznik wydajności) na dowolnym poziomie uprzywilejowania. Używane są dwa liczniki wydajności: do mierzenia czasu trwania specyficznego rodzaju zdarzeń oraz liczby wystąpień takich zdarzeń.

Rejestry MMX

W podrozdziale 10.3 wspominaliśmy, że w technologii MMX Pentium używa się kilku rodzajów danych 64-bitowych. W rozkazach MMX są stosowane 3-bitowe pola adresów rejestrów, jest więc możliwe obsługiwanie ośmiu rejestrów MMX. W rzeczywistości w procesorze nie występują specjalne rejestry MMX. Zamiast tego procesor korzysta z metody koordynowania nazw (*aliasing*), przedstawionej na rys. 12.22. Do przechowywania wartości MMX służą istniejące rejestry zmiennopozycyjne. W szczególności wykorzystuje się 64 bity niższego rzędu (mantysa) każdego z rejestrów zmiennopozycyjnych do tworzenia ośmiu rejestrów MMX. Zatem istniejąca architektura Pentium może być poszerzona w celu obsługiwania technologii MMX. Podstawowe właściwości wykorzystania tych rejestrów dla potrzeb MMX są następujące:

- Przypomnijmy sobie, że rejestry zmiennopozycyjne są traktowane jako stos służący operacjom zmiennopozycyjnym. W operacjach MMX rejestry te mogą być dostępne bezpośrednio.



Rysunek 12.22. Odwzorowanie rejestrów MMX na rejestry zmiennopozycyjne

- Gdy po dowolnych operacjach zmiennopozycyjnych jest wykonywany rozkaz MMX, słowo znacznika FP jest oznaczane jako ważne. Odzwierciedla to przejście od działania w trybie stosu do bezpośredniego adresowania rejestrów.
- Rozkaz EMMS (*Empty MMX State* – rejestry MMX puste) ustawia bity słowa znacznika FP w celu wskazania, że wszystkie rejestry są puste. Ważne jest, aby programista wstawiał ten rozkaz na końcu bloku kodów MMX, żeby następujące po nim operacje zmiennopozycyjne przebiegały prawidłowo.
- Gdy w rejestrze MMX zostanie zapisana wartość, wszystkie bity [79÷64] odpowiedniego rejestru FP (bity znaku i wykładnika) są ustawiane jako 1. Powoduje to ustawienie wartości w rejestrze FP jako NaN (nie liczba) lub nieskończoność, jeśli wartość ta jest traktowana jako zmiennopozycyjna. Dzięki temu wartość danych MMX nie będzie wyglądała jako ważna wartość zmiennopozycyjna.

Przetwarzanie przerwań

Przetwarzanie przerwań wewnątrz procesora jest udogodnieniem przewidzianym do wspierania systemu operacyjnego. Umożliwia ono zawieszenie programu użytkowego w celu obsłużenia różnych warunków przerwań oraz późniejsze wznowienie programu.

Przerwania i wyjątki

Dwie klasy zdarzeń skłaniają Pentium do zawieszenia wykonywania bieżącego strumienia rozkazów i do reakcji na te zdarzenia: przerwania i wyjątki. W obu przypadkach procesor zachowuje kontekst bieżącego procesu i przechodzi do wcześniej określonej procedury związanej z obsługą warunku. *Przerwanie* jest generowane przez sygnał pochodzenia sprzętowego i może nastąpić w dowolnym czasie podczas realizacji programu. *Wyjątek* jest generowany przez oprogramowanie i jest spowodowany wykonywaniem rozkazu. Istnieją dwa źródła przerwań i dwa źródła wyjątków:

1. Przerwania:
   - **Maskowane.** Odebrane poprzez końcówkę INTR procesora. Procesor nie rozpoznaje przerwania maskowanego, chyba że jest ustawiony znacznik zezwolenia przerwania (IF).
   - **Niemaskowane.** Odebrane poprzez końcówkę NMI procesora. Rozpoznaniu takich przerwań nie można zapobiec.
2. Wyjątki:
   - **Wykryte przez procesor.** Są wynikiem napotkania błędu przez procesor usiłujący wykonać rozkaz.
   - **Programowane.** Istnieją rozkazy generujące wyjątki (INT0, INT3, INT i BOUND).

Tablica wektorów przerwań

Przetwarzanie przerwań w Pentium opiera się na używaniu tablicy wektorów przerwań. Każdemu rodzajowi przerwania jest przypisany numer, służący do indeksowania w tablicy wektorów przerwań. Tablica ta zawiera 256 32-bitowych wektorów przerwań, które są adresami (segment i wyrównanie) procedur obsługi przerwania dla określonego numeru przerwania.

Tabela 12.2. Tabela wektorów wyjątków i przerwań procesora Pentium

Numer wektora	Opis
0	Błąd dzielenia; przepełnienie dzielenia lub dzielenie przez zero.
1	Wyjątek związany z uruchamianiem programu; obejmuje różne błędy i pułapki związane z uruchamianiem (*debugging*).
2	Przerwanie za pośrednictwem końcówki NMI; sygnał na końcówce NMI.
3	Punkt przerwania; spowodowany przez rozkaz INT 3, będący 1-bajtowym rozkazem używanym przy uruchamianiu.
4	Przepełnienie wykryte przez INTO; następuje, gdy procesor wykonuje INTO przy ustawionym znaczniku stanu OF.
5	Przekroczenie zakresu BOUND; rozkaz BOUND powoduje porównanie zawartości rejestru z granicami przechowywanymi w pamięci i wygenerowanie przerwania, jeśli zawartość rejestru wykracza poza granice.
6	Nieokreślony kod operacji.
7	Urządzenie nieosiągalne; próba użycia rozkazu ESC lub WAIT nie powiodła się z powodu braku urządzenia zewnętrznego.
8	Podwójny błąd; podczas tego samego rozkazu następują dwa przerwania, które nie mogą być obsługiwane szeregowo.
9	Zarezerwowany.
10	Nieważny segment stanu zadania; segment opisujący żądane zadanie nie został zainicjowany lub jest nieważny.
11	Segment nieobecny; wymagany segment nie jest obecny.
12	Błąd stosu; przekroczenie wartości granicznej segmentu stosu lub segment stosu nie jest obecny.
13	Ogólna ochrona; naruszenie ochrony, które nie powoduje innych wyjątków (np. zapisywania w segmencie przeznaczonym tylko do odczytu).
14	Błąd strony.
15	Zarezerwowany.
16	Błąd zmiennopozycyjny; generowany przez rozkaz arytmetyki zmiennopozycyjnej.
17	Sprawdzenie wyrównania; dostęp do słowa przechowywanego pod nieparzystym adresem bajtowym lub do podwójnego słowa, którego adres nie jest wielokrotnością 4.
18	Sprawdzenie komputera; zależny od modelu.
19÷31	Zarezerwowane.
32÷255	Wektory przerwań użytkownika; dostarczane, gdy jest aktywowany sygnał INTR.

Niezacienione – wyjątki, zacienione – przerwania.

W tabeli 12.2 jest podane przypisanie numerów w tablicy wektorów przerwań; zapisy zacieniowane dotyczą przerwań, a niezacieniowane – wyjątków. Przerwanie sprzętowe NMI jest rodzaju 2. Przerwaniom sprzętowym INTR przypisano numery od 32 do 255. Gdy jest generowane przerwanie INTR, na magistrali musi mu towarzyszyć numer wektora przerwania odnoszący się do tego przerwania. Pozostałe numery wektorów dotyczą wyjątków.

Jeśli nie rozstrzygnięte jest więcej niż jedno przerwanie lub wyjątek, procesor obsługuje je w przewidywalnej kolejności. Położenie numeru wektora wewnątrz tablicy nie odzwierciedla priorytetu. Priorytety wyjątków i przerwań są zorganizowane w postaci pięciu klas. Oto one, w kolejności malejącego priorytetu:

- **Klasa 1.** Pułapka w poprzednim rozkazie (wektor nr 1).
- **Klasa 2.** Przerwania zewnętrzne (2, 32÷255).
- **Klasa 3.** Błędy pobierania następnego rozkazu (3, 14).
- **Klasa 4.** Błędy dekodowania następnego rozkazu (6, 7).
- **Klasa 5.** Błędy wykonywania rozkazu (0, 4, 5, 8, 10÷14, 16, 17).

Obsługa przerwań

Podobnie jak przy przeniesieniu wykonywania za pomocą rozkazu CALL, przy przeniesieniu do procedury obsługi przerwań jest wykorzystywany stos systemowy w celu zachowania stanu procesora. Gdy następuje przerwanie i jest ono rozpoznane przez procesor, ma miejsce następująca sekwencja zdarzeń:

1. Jeśli transfer powoduje zmianę poziomu uprzywilejowania, to rejestr bieżącego segmentu stosu i rejestr bieżącego rozszerzonego wskaźnika stosu (ESP) są umieszczane na stosie.
2. Bieżąca wartość rejestru EFLAGS jest umieszczana na stosie.
3. Znaczniki stanu przerwania (IF) i pułapki (TF) są zerowane. Blokuje to przerwania INTR i pułapkę lub postępowanie w trybie krokowym.
4. Bieżący wskaźnik segmentu kodu (CS) i bieżący wskaźnik rozkazu (IP lub EIP) są umieszczane na stosie.
5. Jeśli przerwaniu towarzyszy kod błędu, to kod błędu jest umieszczany na stosie.
6. Zawartość wektora przerwania jest pobierana i ładowana do rejestrów CS i IP lub EIP. Wykonywanie rozkazów przez procesor jest kontynuowane począwszy od procedury obsługi przerwania.

W celu wznowienia pracy po przerwaniu w ramach procedury obsługi przerwania jest wykonywany rozkaz IRET. Powoduje to odtworzenie wszystkich wartości umieszczonych na stosie; wykonywanie programu jest wznawiane począwszy od punktu przerwania.

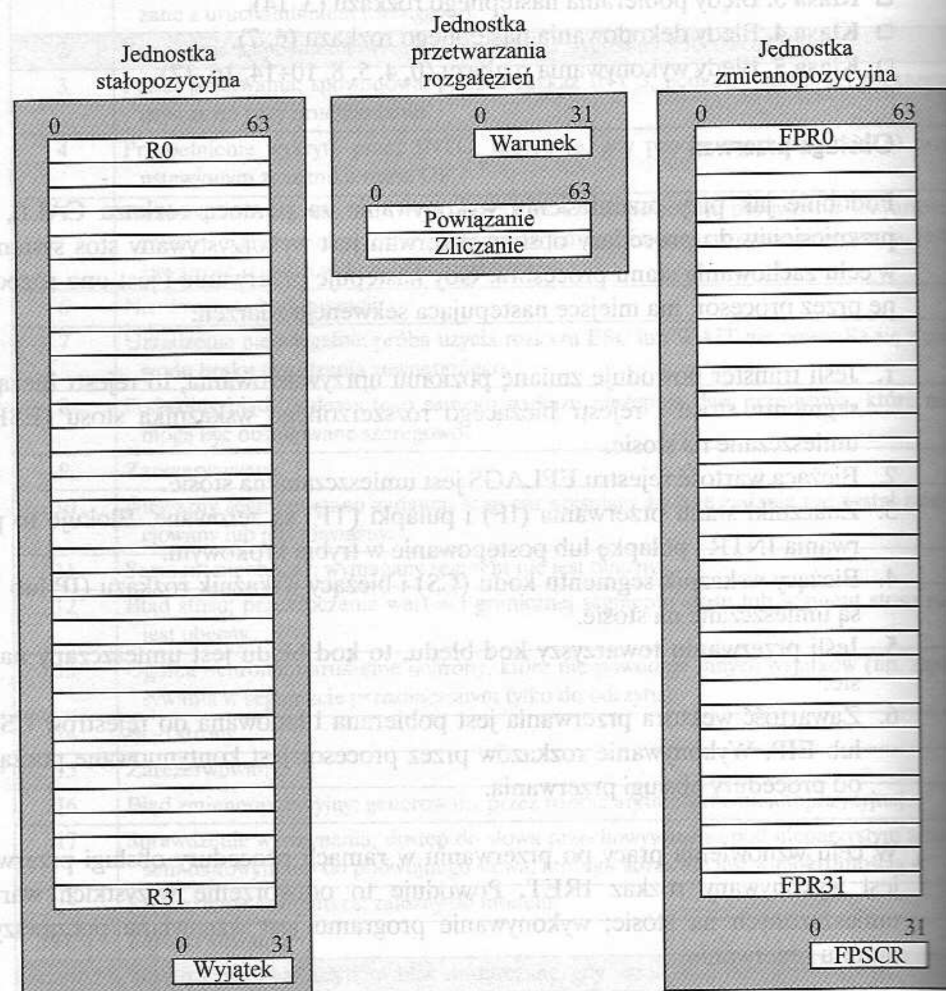
12.6. Procesor PowerPC

Organizacja procesora PowerPC była przedstawiona na rys. 4.14. W tym podrozdziale przeanalizujemy niektóre rozwiązania szczegółowe dotyczące wersji 64-bitowej.

Organizacja rejestrów

Na rysunku 12.23 są pokazane rejestry PowerPC widzialne dla użytkownika. Jednostka stałopozycyjna zawiera następujące rejestry:

- **Ogólnego przeznaczenia.** Procesor zawiera trzydzieści dwa 64-bitowe rejestry ogólnego przeznaczenia (robocze). Mogą one być używane do ładowania, zapisu i manipulowania argumentami oraz jako rejestry adresowania pośredniego. Rejestr 0



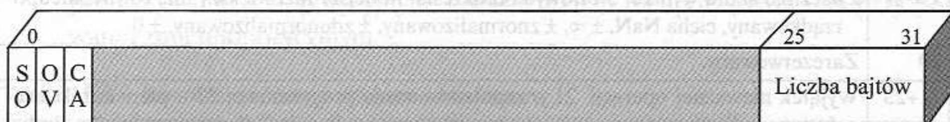
Rysunek 12.23. Rejestry procesora PowerPC widzialne dla użytkownika

jest traktowany nieco odmiennie. W przypadku operacji ładowania i zapisu oraz niektórych rozkazów dodawania rejestr 0 jest traktowany jako stale wyzerowany, niezależnie od swojej rzeczywistej zawartości.

- **Rejestr wyjątku** (**XER** – *Exception Register*). Zawiera 3 bity zgłaszania wyjątków w operacjach arytmetycznych na liczbach całkowitych. Zawiera również pole zliczania bajtów używane jako argument w przypadku niektórych rozkazów dotyczących łańcuchów (rys. 12.23a).

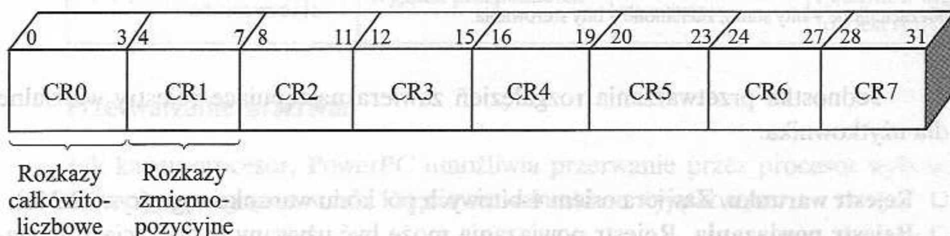
Jednostka zmiennopozycyjna zawiera dodatkowe rejestry widzialne dla użytkownika:

- **Ogólnego przeznaczenia (robocze).** Są to 32 64-bitowe rejestry robocze używane do wszystkich operacji zmiennopozycyjnych.
- **Zmiennopozycyjny rejestr stanu i sterowania** (**FPSCR** – *Floating-Point Status and Control Register*). Ten 32-bitowy rejestr zawiera bity sterujące pracą jednostki zmiennopozycyjnej oraz bity rejestrujące stan wynikający z operacji zmiennopozycyjnych (tabela 12.3).



SO – przepełnienie sumaryczne (*summary overflow*): ustawiany na 1 w celu wskazania przepełnienia występującego podczas wykonywania rozkazu; pozostaje równy 1, aż do wyzerowania programowego
OV – przepełnienie (*overflow*): ustawiany na 1 w celu wskazania przepełnienia podczas wykonywania rozkazu; zerowany przez następny rozkaz, jeśli nie ma przepełnienia
CA – przeniesienie (*carry*): ustawiany na 1 w celu wskazania przeniesienia na zewnątrz bitu 0 podczas wykonywania rozkazu
Liczba bajtów – określa liczbę bajtów, które mają być przeniesione przez rozkaz ładowania/zapisu łańcucha

(a) Stałopozycyjny rejestr wyjątku (XER)



(b) Rejestr warunku

Rysunek 12.24. Formaty rejestrów procesora PowerPC

Tabela 12.3. Zmiennopozycyjny rejestr stanu i sterowania procesora PowerPC

Bit	Definicja
0	Podsumowanie wyjątku. Ustawiany, gdy następuje jakikolwiek wyjątek; pozostaje ust wiony, aż do wyzerowania przez program.
1	Podsumowanie dozwolonego wyjątku. Ustawiany, gdy następuje jakikolwiek dozwolo wyjątek.
2	Podsumowanie wyjątku nieważnej operacji. Ustawiany, jeśli nastąpił wyjątek spowod wany przez nieważną operację.
3	Wyjątek przepełnienia. Wielkość wyniku przekracza tę, która może być reprezentowan
4	Wyjątek niedomiaru. Wynik jest zbyt mały, aby mógł być znormalizowany.
5	Wyjątek dzielenia przez zero. Dzielnik jest zerem, a dzielna jest skończona i niezerowa
6	Wyjątek niedokładności. Zaokrąglony wynik różni się od wyniku pośredniego lub nast puje przepełnienie przy zablokowanym wyjątku przepełnienia.
7÷12	Wyjątek nieważnej operacji: 7: sygnalizująca NaN; 8: $(\infty-\infty)$); 9: $(\infty\div\infty)$; 10: $(0\div0$ 11: $(\infty\times0)$; 12: porównanie obejmujące NaN.
13	Zaokrąglenie ułamka. Zaokrąglenie wyniku inkrementowało ułamek.
14	Niedokładność ułamka. Zaokrąglenie wyniku zmieniło ułamek lub wystąpiło przepełni nie przy zablokowanym wyjątku przepełnienia.
15÷19	Znaczniki stanu wyniku. 5-bitowy kod określa: mniejszy niż, większy niż, równy, nieup rządkowany, cicha NaN, $\pm\infty$, $\pm$ znormalizowany, $\pm$ zdenormalizowany, $\pm$ 0.
20	Zarezerwowany.
21÷23	Wyjątek nieważnej operacji. 21 – zapotrzebowanie programowe; 22 – pierwiastek kw dratowy z liczby ujemnej; 23 – konwersja liczby całkowitej obejmująca wielką liczb nieskończoność lub NaN.
24	Zezwolenie wyjątku nieważnej operacji.
25	Zezwolenie wyjątku przepełnienia.
26	Zezwolenie wyjątku niedomiaru.
27	Zezwolenie wyjątku dzielenia przez zero.
28	Zezwolenie wyjątku niedokładności.
29	Tryb niezgodny z IEEE.
30÷31	Sterowanie zaokrąglaniem. 2-bitowy kod określa: do najbliższej, w stronę zera, w stro $+\infty$, w stronę $-\infty$.

Niezacienione – bity stanu; zacienione – bity sterowania.

Jednostka przetwarzania rozgałęzień zawiera następujące rejestry widzia dla użytkownika:

- **Rejestr warunku.** Zawiera osiem 4-bitowych pól kodu warunkowego (rys. 12.24b
- **Rejestr powiązania.** Rejestr powiązania może być używany w operacjach roz łęzienia warunkowego do adresowania pośredniego adresu docelowego. M być również używany do operacji wywołania i powrotu. Jeżeli jest ustawion LK w rozkazie rozgałęzienia warunkowego, to adres następujący po rozka rozgałęzienia jest umieszczany w rejestrze powiązania i może być użyty przy p niejszym powrocie.

- **Rejestr zliczania.** Rejestr zliczania może być używany do sterowania pętlą, co wyjaśniliśmy w rozdz. 10; jego stan jest zmniejszany za każdym razem, gdy jest on sprawdzany w ramach rozkazu rozgałęzienia warunkowego. Innym zastosowaniem tego rejestru jest pośrednie adresowanie adresu docelowego w operacjach rozgałęzienia.

Pola rejestru warunku mają wiele zastosowań. Pierwsze cztery bity (CR0) są ustawiane dla wszystkich operacji arytmetycznych na liczbach całkowitych, dla których jest ustawiony bit Rc. Jak wynika z tabeli 12.4, pole to wskazuje, czy wynik operacji jest dodatni, ujemny bądź zerowy. Czwarty bit jest kopią bitu łącznego przepełnienia z rejestru XER. Następne pole (CR1) jest ustawiane dla wszystkich zmiennopozycyjnych operacji arytmetycznych, dla których jest ustawiony bit Rc. W tym przypadku 4 bity są ustawiane jako równe pierwszym 4 bitom FPSCR (tab. 12.3). Wreszcie 8 pól warunku (CR0÷CR7) może być używanych w przypadku rozkazu porównania; zawartość pola jest określana w samym rozkazie. W odniesieniu do stało- i zmiennopozycyjnych rozkazów porównania, pierwsze 3 bity wyznaczonego pola warunku określają, czy pierwszy argument jest mniejszy, większy lub równy drugiemu argumentowi. Czwarty bit jest bitem łącznego przepełnienia przy porównywaniu stałopozycyjnym, a wskaźnikiem nieuporządkowania przy porównywaniu zmiennopozycyjnym.

Tabela 12.4. Interpretacja bitów w rejestrze warunkowym

Pozycja bitu	CR0 (rozkaz całkowitoliczbowy przy Rc=1)	CR1 (rozkaz zmiennopozycyjny przy Rc=1)	CRi (stałopozycyjny rozkaz porównania)	CRi (zmiennopozycyjny rozkaz porównania)
i	Wynik < 0	Podsumowanie wyjątku	op1 < op2	op1 < op2
$i+1$	Wynik > 0	Podsumowanie dozwolonego wyjątku	op1 > op2	op1 > op2
$i+2$	Wynik=0	Podsumowanie wyjątku nieważnej operacji	op1=op2	op1=op2
$i+3$	Przepełnienie podsumowania	Wyjątek przepełnienia	Przepełnienie podsumowania	Nieuporządkowany (jednym z argumentów jest NaN)

Przetwarzanie przerwań

Jak każdy procesor, PowerPC umożliwia przerwanie przez procesor wykonywania bieżącego programu w celu zajęcia się warunkiem wyjątkowym.

Rodzaje przerwań

Przerwania w PowerPC są podzielone na wywołane przez pewien warunek lub zdarzenie systemowe oraz na spowodowane przez wykonywanie rozkazu. W tabeli 12.5 są wymienione przerwania rozpoznawane przez PowerPC.

Tabela 12.5. Tabela przerwań PowerPC

Punkt wejścia	Rodzaj przerwania	Opis
00000h	Zarezerwowane	
00100h	Resetowanie systemu	Potwierdzenie twardych lub miękkich sygnałów wejściowych resetowania przez zewnętrzne układy logiczne.
00200h	Sprawdzenie komputera	Potwierdzenie TEA# procesorowi, gdy dozwolone jest rozpoznawanie operacji sprawdzenia komputera.
00300h	Przechowywanie danych	Przykłady: błąd strony danych; naruszenie praw dostępu w czasie ładowania/zapisu.
00400h	Przechowywanie rozkazów	Błąd strony programu; próba pobrania rozkazu z segmentu wejścia-wyjścia; naruszenie praw dostępu.
00500h	Zewnętrzne	Potwierdzenie sygnału wejściowego zewnętrznego przerwania procesora przez zewnętrzne układy logiczne, gdy rozpoznanie zewnętrznego przerwania jest dozwolone.
00600h	Wyrównanie	nieudana próba dostępu do pamięci spowodowana przez niewyrównany argument.
00700h	Programowe	Przerwanie zmiennopozycyjne; użytkownik próbuje wykonać uprzywilejowany rozkaz; wykonywany jest rozkaz pułapki przy spełnionym określonym warunku; nielegalny rozkaz.
00800h	Przetwarzanie zmiennopozycyjne nieosiągalne	Próba wykonania rozkazu zmiennopozycyjnego przy zablokowanej jednostce zmiennopozycyjnej.
00900h	Rejestr odejmujący	Wyczerpanie możliwości rejestru odejmującego (*decrementer register*), gdy dozwolone jest rozpoznawanie zewnętrznego przerwania.
00A00h	Zarezerwowane	
00B00h	Zarezerwowane	
00C00h	Wywołanie systemowe	Wykonywanie rozkazu wywołania systemowego.
00D00h	Śledzenie	Jednoetapowe lub rozgałęzieniowe przerwanie śledzenia.
00E00h	Wspomaganie zmiennopozycyjne	Próba wykonania stosunkowo rzadkiej, złożonej operacji zmiennopozycyjnej (np. operacji na liczbie zdenormalizowanej).
00E10h÷÷00FFFh	Zarezerwowane	
01000h÷÷02FFFh	Zarezerwowane (specyficzne wdrożenie)	

Niezacienione – przerwania spowodowane przez wykonywanie rozkazu, zacienione – przerwania nie spowodowane przez wykonywanie rozkazu.

Większość przerwań wymienionych w tabeli jest łatwa do zrozumienia. Kilka wymaga dodatkowego komentarza. Systemowe przerwanie ponownego inicjowania pracy zdarza się przy włączaniu zasilania oraz po naciśnięciu przycisku ponownego inicjowania pracy (*reset*). Powoduje ono powtórne zainicjowanie (*reboot*

systemu. Maszynowe przerwanie kontrolne dotyczy pewnych anomalii, takich jak błąd parzystości pamięci podręcznej lub odniesienie do nieistniejącej lokacji pamięci. Może ono spowodować przejście systemu do tzw. stanu zatrzymania kontrolnego (*check-stop state*). W tym stanie zawieszone są operacje procesora i jest zamrożona zawartość rejestrów aż do powtórnego zainicjowania systemu. Przerwanie wspomagania zmiennopozycyjnego pozwala procesorowi na wywołanie procedur programowych w celu zakończenia operacji, które nie mogą być przeprowadzone przez jednostkę zmiennopozycyjną, w tym operacji na liczbach zdenormalizowanych, lub operacji określanych przez niewdrożone kody operacji zmiennopozycyjnych.

Rejestr stanu maszyny (MSR)

Podstawowe znaczenie przy przerwaniu programu ma zdolność do odtworzenia stanu procesora w czasie zaistnienia przerwania. Dotyczy to nie tylko zawartości różnych rejestrów, ale także różnych warunków sterowania związanych z realizacją programu. Warunki te są zwykle sumowane w MSR (tabela 12.6). Kilka bitów tego rejestru wymaga dodatkowego komentarza.

Tabela 12.6. Rejestr stanu procesora PowerPC

Bit	Definicja
0	Procesor jest w trybie 32-bitowym/64-bitowym.
1÷44	Zarezerwowane.
45	Zarządzanie mocą dozwolone/zablokowane.
46	Zależne od wdrożenia.
47	Określa, czy programy obsługi przerwań funkcjonują w trybie najpierw najmłodszy bajt, czy najpierw najstarszy bajt.
48	Przerwanie zewnętrzne dozwolone/zablokowane.
49	Stan uprzywilejowany/nieuprzywilejowany.
50	Jednostka zmiennopozycyjna osiągalna/nieosiągalna.
51	Przerwania kontroli komputera dozwolone/zablokowane.
52	Tryb 0 wyjątku zmiennopozycyjnego.
53	Śledzenie jednoetapowe dozwolone/zablokowane.
54	Śledzenie rozgałęzieniowe dozwolone/zablokowane.
55	Tryb 1 wyjątku zmiennopozycyjnego.
56	Zarezerwowane.
57	Najbardziej znaczącą częścią adresu wyjątku jest 000h/FFFh.
58	Translacja adresu rozkazu włączona/wyłączona.
59	Translacja adresu danych włączona/wyłączona.
60÷61	Zarezerwowane.
62	Przerwanie jest przywracalne/nieprzywracalne.
63	Procesor jest w trybie najpierw najstarszy bajt/najpierw najmłodszy bajt.

Niezacienione – kopiowane do SRR1, zacienione – nie kopiowane do SRR1.

Gdy jest ustawiony bit trybu uprzywilejowania (bit 49), procesor pracuje na poziomie uprzywilejowania użytkownika. Dostępny jest tylko pewien podzbiór listy rozkazów. Gdy bit ten jest wyzerowany, procesor pracuje na poziomie uprzywilejowania nadzorcy. Umożliwia to dostęp do wszystkich rozkazów oraz do niektórych rejestrów systemowych (takich jak MSR) niedostępnych z poziomu użytkownika.

Wartości dwóch bitów wyjątku zmiennopozycyjnego (bity 52 i 55) określają rodzaje przerwań, które mogą być generowane przez jednostkę zmiennopozycyjną. Ich interpretacja jest następująca:

FE0	FE1	Przerwania, które zostaną rozpoznane
0	0	żadne
0	1	nieprecyzyjne, nieprzywracalne
1	0	nieprecyzyjne, przywracalne
1	1	precyzyjne

Gdy jest ustawiony bit śledzenia krokowego (bit 53), procesor wraca do śledzącego programu obsługi przerwań po udanym wykonaniu każdego rozkazu. Natomiast gdy jest ustawiony bit śledzenia rozgałęzień (bit 54), procesor wraca do śledzącego programu obsługi przerwań po udanym wykonaniu każdego rozkazu rozgałęzienia, niezależnie od tego, czy rozgałęzienie nastąpiło.

Bit translacji adresu rozkazu (bit 58) oraz bit translacji adresu danych (bit 59) określają, czy jest używane rzeczywiste adresowanie, czy też jednostka zarządzania pamięcią prowadzi translację adresu.

Obsługa przerwań

Gdy następuje przerwanie i jest ono rozpoznane przez procesor, ma miejsce następująca sekwencja zdarzeń:

1. Procesor umieszcza adres następnego rozkazu przewidzianego do wykonania w rejestrze zachowania/odtworzenia 0 (SRR0). Jest to adres właśnie wykonywanego rozkazu, jeśli przerwanie zostało spowodowane przez nieudaną próbę wykonania tego rozkazu; w przeciwnym razie jest to adres następnego rozkazu przewidzianego do wykonania po rozkazie bieżącym.
2. Procesor kopiuje informację o stanie z rejestru MSR do rejestru zachowania/odtworzenia 1 (SRR1). Kopiowane są bity, które w tabeli 12.6 nie są zacieniowane. Do pozostałych bitów SRR1 jest ładowana informacja specyficzna dla rodzaju przerwania.
3. W rejestrze MSR jest ustawiana wartość określona sprzętowo, specyficzna dla rodzaju przerwania. Dla wszystkich rodzajów przerwań translacja adresu jest wyłączona, a przerwania zewnętrzne są zablokowane.
4. Następnie procesor przekazuje sterowanie odpowiedniemu programowi obsługi przerwań. Adresy programów obsługi przerwań są przechowywane w tabeli przerwań (tabeli 12.5). Adres podstawowy tej tabeli jest wyznaczany przez bit 57 w rejestrze MSR.

Aby wrócić po przerwaniu, w ramach procedury obsługi przerwań jest wykonywany rozkaz rfi (*return from interrupt*). Powoduje on, że bity zachowane w rejestrze SRR1 są odtwarzane w rejestrze MSR. Wykonywanie jest wznawiane począwszy od lokacji przechowywanej w SRR0.

12.7. Polecana literatura

[PATT01] i [MOSH01] zawierają doskonałe omówienie zagadnień przetwarzania potokowego przedstawionych w tym rozdziale. Prace [HENN91] i [HVAN93] zawierają szczegółowy opis przetwarzania potokowego. W [SOHI90] można znaleźć doskonałe, szczegółowe omówienie problemów projektowania sprzętu związanych z potokowym przetwarzaniem rozkazów.

W pracy [EVER01] przeanalizowano ewolucję strategii przewidywania rozgałęzień. W [CRAG92] znajduje się szczegółowe studium przewidywania rozgałęzień w potokach rozkazów. W [DUBE91] i [LILJ88] są przeanalizowane różne strategie przewidywania rozgałęzień, które mogą być używane w celu zwiększania wydajności potokowego przetwarzania rozkazów. W [KAEL91] są przeanalizowane utrudnienia w przewidywaniu rozgałęzień powodowane przez rozkazy, których adres docelowy jest zmienny.

Potokowe przetwarzanie rozkazów w procesorze Intel 80486 jest opisane w [TABA91]. W [BREY00] dobrze opisano przetwarzanie przerwań w Pentium, a w [SHAN95] – w PowerPC.

BREY00 Brey B.: *The Intel 32-bit Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro and Pentium II Processors*. Upper Saddle River, Prentice Hall, 2000.

CRAG92 Cragon H.: *Branch Strategy Taxonomy and Performance Models*. Los Alamitos, IEEE Computer Society Press, 1992.

DUBE91 Dubey P., Flynn M.: „Branch Strategies: Modeling and Optimization". *IEEE Transactions on Computers*, October 1991.

EVER01 Evers M., Yeh T.: „Understanding Branches and Designing Branch Predictors for High-Performance Microprocessors". *Proceedings of the IEEE*, November 2001.

HENN91 Hennessy J., Jouppi N.: „Computer Technology and Architecture: An Evolving Interaction". *Computer*, September 1991.

HWAN93 Hwang K.: *Advanced Computer Architecture*. New York, McGraw-Hill, 1993.

KAEL91 Kaeli D., Emma P.: „Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns". *Proceeding, 18th Annual International Symposium on Computer Architecture*, May 1991.

LILJ88 Lilja D.: „Reducing the Branch Penalty in Pipelined Processors". *Computer*, July 1988.

MOSH01 Moshovos A., Sohi G.: „Microarchitectural Innovations: Boosting Microprocessor Performance Beyond Semiconductor Technology Scaling". *Proceedings of the IEEE*, November 2001.

PATT01 Patt Y.: „Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution". *Proceedings of the IEEE*, November 2001.

SHAN95 Shanley T.: *PowerPC System Architecture*. Reading, Addison-Wesley, 1995.

SOHI90 Sohi G.: „Instruction Issue Logic for High-Performance Interruptable, Multiple Functional Unit, Pipelined Computers". *IEEE Transaction on Computers*, March 1990.

TABA91 Tabak D.: *Advanced Microprocessors*. New York, McGraw-Hill, 1991.

12.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Cykl rozkazu – *instruction cycle*
Kod warunkowy – *condition code*
Potok rozkazów – *instruction pipeline*
Przewidywanie rozgałęzień – *branch prediction*
Rozgałęzienie opóźnione – *delayed branch*
Słowo stanu programu (PSW) – *program status word*
Wstępne pobieranie rozkazów – *instruction prefetch*
Znacznik – *flag*

Pytania kontrolne

12.1. Jaką ogólną rolę pełnią rejestry procesora?
12.2. Jakie kategorie danych są zwykle obsługiwane przez rejestry widzialne dla użytkownika?
12.3. Jaka jest funkcja kodów warunkowych?
12.4. Co to jest słowo stanu programu?
12.5. Dlaczego zastosowanie dwuetapowego potoku rozkazów prawdopodobnie nie jest w stanie skrócić cyklu rozkazu do połowy w porównaniu z sytuacją, w której nie zastosowano żadnego przetwarzania potokowego?
12.6. Wymień i krótko objaśnij różne sposoby postępowania z rozkazami rozgałęzienia warunkowego w potoku rozkazów.
12.7. W jaki sposób bity historii są wykorzystywane do przewidywania rozgałęzień?

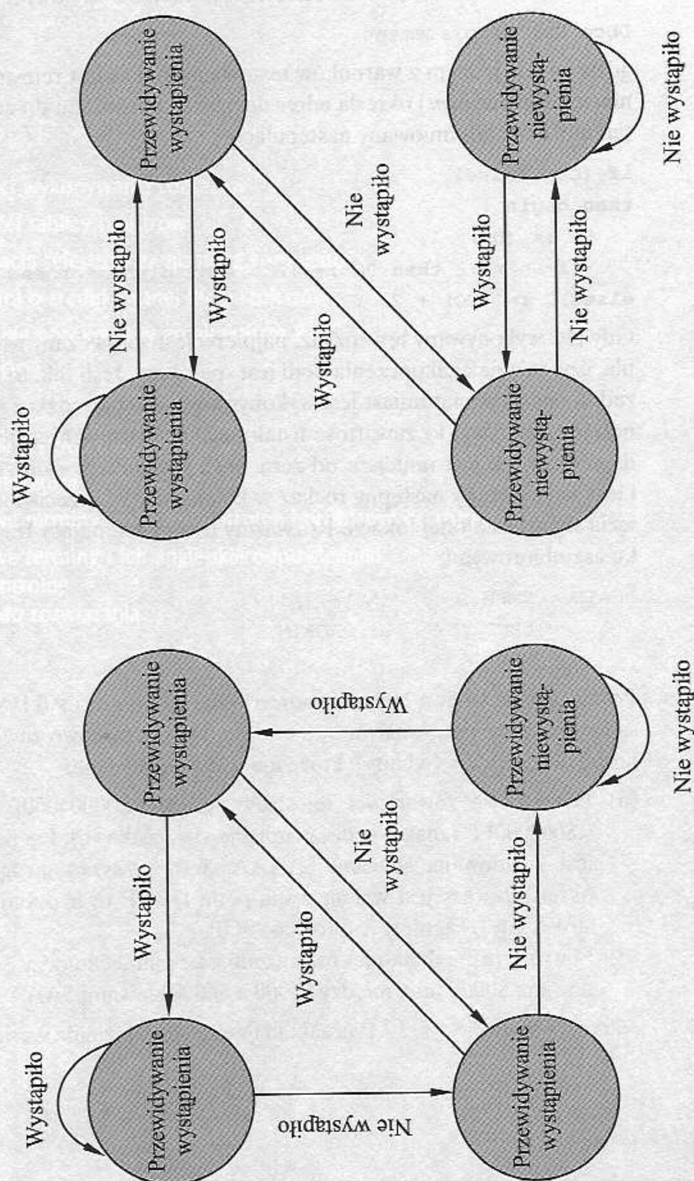
Problemy do rozwiązania

12.1. (a) Jeśli ostatnią operacją przeprowadzoną w komputerze o słowie 8-bitowym było dodawanie, w którym dwoma argumentami były 2 i 3, to jaka jest wartość następujących znaczników stanu (flag)?
- przeniesienie;
- zero;
- przepełnienie;
- znak;
- parzystość;
- półprzeniesienie.

(b) Jakie byłyby te wartości, jeśli argumentami byłyby –1 (uzupełnienie do dwóch) i + 1.

12.2. Rozważmy diagram czasowy na rys. 12.10. Załóżmy, że mamy do czynienia z potokiem dwuetapowym (pobieranie, wykonanie). Narysuj nowy diagram w celu wykazania, ile jednostek czasu potrzeba do wykonania 4 rozkazów.

12.3. Rozważ sekwencję rozkazów o długości n znajdującą się w potoku rozkazów. Niech p będzie prawdopodobieństwem napotkania rozkazu rozgałęzienia warunkowego lub bezwarunkowego i niech q będzie prawdopodobieństwem, że wykonanie rozkazu rozgałęzienia I powoduje skok pod adres, który nie jest kolejnym adresem. Przyjmij, że każdy taki skok wymaga oczyszczenia potoku, zniszczenia bieżącego przetwarzania rozkazów po pojawieniu się I na końcu ostatniego etapu. Zmodyfikuj równania 12.1 i 12.2 tak, aby uwzględnić te prawdopodobieństwa.



Rysunek 12.25. Graf stanów przetwarzania rozgałęzień (problem 12.5)

12.4. Jednym z ograniczeń rozwiązania wielostrumieniowego obsługi rozgałęzień w potoku jest to, że napotykane są dodatkowe rozgałęzienia, zanim pierwsze rozgałęzienie jest wykonane do końca. Zasugeruj dwa dodatkowe ograniczenia lub niedogodności.

12.5. Rozważ wykresy stanów z rys. 12.25.

(a) Opisz funkcjonowanie każdego z nich.

(b) Porównaj je z wykresem stanów odnoszącym się do przewidywania rozgałęzień w podrozdz. 12.4. Omów mocne i słabe strony każdego z tych trzech rozwiązań przewidywania rozgałęzień.

12.6. Procesory Motorola 680x0 zawierają rozkaz „dekrementuj i rozgałęziaj zależnie od warunku" (*Decrement and Branch According to Condition*), który ma następującą postać:

```
Dbcc Dn, displacement
```

gdzie cc jest jednym z warunków testowalnych, Dn jest rejestrem roboczym, a przesunięcie (*displacement*) określa adres docelowy w stosunku do adresu bieżącego. Rozkaz ten może być zdefiniowany następująco:

```
if (cc = False)
then begin
    Dn := (Dn) -1;
    if Dn ≠ -1 then PC := (PC) + displacement end
else PC := (PC) + 2;
```

Gdy jest wykonywany ten rozkaz, najpierw jest sprawdzany warunek w celu stwierdzenia, czy warunek zakończenia pętli jest spełniony. Jeśli tak, to nie jest przeprowadzana żadna operacja, natomiast jest wykonywany następny rozkaz w kolejności. Jeśli warunek jest fałszywy, to zawartość ustalonego rejestru danych jest zmniejszana i sprawdzana, czy nie jest mniejsza od zera. Jeśli jest mniejsza od zera, pętla jest kończona i jest wykonywany następny rozkaz w kolejności. W przeciwnym razie program rozgałęzia się do ustalonej lokacji. Rozważmy teraz następujący fragment programu w języku asemblerowym:

```
AGAIN   CMPM.L      (A0)+,(A1)+
        DBNE        D1, AGAIN
        NOP
```

Porównywane są dwa łańcuchy adresowane za pomocą A0 i A1; wskaźniki łańcuchów są zmniejszane przy każdym odniesieniu. D1 początkowo zawiera wartość określającą liczbę długich słów (4 bajty), które mają być porównane.

(a) Początkowe zawartości rejestrów są: A0 = \$00004000, A1 = \$00005000 i D1 = = \$000000FF (znak \$ oznacza notację szesnastkową). Pamięć między \$4000 a \$6000 jest załadowana słowami \$AAAA. Jeśli powyższy program jest uruchomiony, określ, ile razy jest wykonywana pętla DBNE oraz podaj zawartości trzech rejestrów, gdy osiągnięty jest rozkaz NOP.

(b) Powtórz (a), jednak tym razem załóż, że pamięć między \$4000 a \$4FEE jest załadowana \$0000 oraz między \$5000 a \$6000 słowami \$AAA.

12.7. Narysuj ponownie rys. 12.19c, zakładając że rozgałęzienie warunkowe nie następuje.

Rozdział 13

Komputery o zredukowanej liście rozkazów

13.1. Właściwości wykonywania rozkazów
13.2. Użycie dużej tablicy rejestrów
13.3. Optymalizacja rejestrów za pomocą kompilatora
13.4. Architektura o zredukowanej liście rozkazów
13.5. Przetwarzanie potokowe w architekturze RISC
13.6. MIPS R4000
13.7. SPARC
13.8. Porównanie architektur RISC i CISC
13.9. Polecana literatura
13.10. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

PODSTAWOWE SPOSTRZEŻENIA

- Badania wykonywania programów napisanych w językach wysokiego poziomu stały się podstawą do zaprojektowania nowego rodzaju architektury procesorów: komputerów o zredukowanej liście rozkazów (*reduced instruction set computer* – RISC). Dominują instrukcje przypisania, co sugeruje, że powinno być zoptymalizowane proste przenoszenie danych. Istnieje również wiele rozkazów IF (jeżeli) i LOOP (pętla), co z kolei sugeruje, że powinny być zoptymalizowane podstawowe mechanizmy sterowania, aby umożliwić efektywne przetwarzanie potokowe. Badania wzorów odwołań do argumentów wskazują, że powinno być możliwe zwiększenie wydajności przez utrzymywanie umiarkowanej ilości argumentów w rejestrach.
- Badania te legły u podstaw najważniejszych własności maszyn RISC: (1) ograniczona lista rozkazów o ustalonym formacie, (2) wielka liczba rejestrów lub zastosowanie kompilatora optymalizującego wykorzystywanie rejestrów oraz (3) nacisk na optymalizację potoku rozkazów.
- Prosta lista rozkazów RISC sama w sobie przyczynia się do efektywnego przetwarzania potokowego, ponieważ na jeden rozkaz przypada mniej operacji i są one bardziej przewidywalne. Architektura listy rozkazów RISC skłania do stosowania techniki rozgałęzień opóźnionych, w której porządek rozkazów rozgałęzienia jest zmieniany wraz z pozostałymi rozkazami, tak aby zwiększyć skuteczność przetwarzania potokowego.

Od czasu powstania komputera z przechowywanym programem, około roku 1950, niewiele było znaczących, prawdziwych innowacji w dziedzinie organizacji i architektury komputerów. Następujące innowacje, chociaż nie tworzące kompletnej listy, stanowią główne osiągnięcia od narodzin komputera:

- **Koncepcja rodziny.** Wprowadzona przez IBM wraz z systemem 360 w roku 1964, powtórzona wkrótce potem przez DEC wraz z PDP-8. Koncepcja rodziny zakłada oddzielenie architektury maszyny od jej implementacji. Oferowany jest zbiór komputerów różniących się stosunkiem ceny do wydajności, prezentujących użytkownikowi tę samą architekturę. Różnice ceny i wydajności wynikają z różnych wdrożeń tej samej architektury.
- **Mikroprogramowana jednostka sterująca.** Zasugerowana przez Wilkesa w roku 1951 i wprowadzona przez IBM w linii S/360 w roku 1964. Mikroprogramowanie ułatwia zadanie zaprojektowania i implementacji jednostki sterującej i wspiera koncepcję rodziny.
- **Pamięć podręczna.** Po raz pierwszy wprowadzona na rynek w modelu 85 IBM S/360 w roku 1968. Dołączenie tego elementu do hierarchii pamięci w istotny sposób poprawia wydajność.
- **Przetwarzanie potokowe.** Sposób wprowadzenia równoległości do w zasadzie sekwencyjnej natury programów złożonych z rozkazów maszynowych. Przykładami są przetwarzanie potokowe rozkazów i przetwarzanie wektorowe.
- **Wieloprocesorowość.** Kategoria ta obejmuje wiele różnych organizacji i celów.

Do tej listy musimy teraz dodać jedną z najbardziej interesujących i (potencjalnie) najważniejszych innowacji: architekturę komputerów o zredukowanej liście rozkazów (RISC). Architektura RISC stanowi drastyczne odchylenie od historycznej tendencji rozwoju architektury procesorów; jest też wyzwaniem wobec konwencjonalnej mądrości wyrażanej w czynach i w słowach przez większość specjalistów od architektury komputerów. Analiza architektury RISC skłania do zastanowienia się nad najważniejszymi problemami organizacji i architektury komputerów.

Chociaż systemy RISC były definiowane i projektowane na różne sposoby przez różne grupy, kluczowymi elementami występującymi w większości projektów są:

- duża liczba rejestrów roboczych lub zastosowanie kompilatorów do optymalizacji wykorzystania rejestrów;
- ograniczony i prosty zbiór rozkazów;
- akcent na optymalizację potoku rozkazów.

W tabeli 13.1 znajduje się porównanie kilku systemów RISC oraz nie-RISC.

Tabela 13.1. Własności niektórych procesorów CISC, RISC i superskalarnych

Dane	Komputery o złożonej liście rozkazów (CISC)			Komputery o zredukowanej liście rozkazów (RISC)		Superskalarne		
	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Rok powstania	1973	1978	1989	1987	1991	1993	1996	1996
Liczba rozkazów	208	303	235	69	94	225		
Rozmiar rozkazu [B]	2÷6	2÷57	1÷11	4	32	4	4	4
Tryby adresowania	4	22	11	1	1	2	1	1
Liczba rejestrów roboczych	16	16	8	40÷520	32	32	40÷520	32
Rozmiar pamięci sterującej [Kb]	420	480	246	–	–	–	–	–
Rozmiar pamięci podręcznej [KB]	64	64	8	32	128	16÷32	32	64

Rozpoczniemy ten rozdział od krótkiego przeglądu pewnych wyników prac nad listami rozkazów, następnie przeanalizujemy każdy z wyżej wymienionych trzech elementów. W dalszym ciągu opiszemy dwa najlepiej udokumentowane projekty RISC.

13.1. Właściwości wykonywania rozkazów

Jednym z najbardziej widocznych elementów ewolucji związanej z komputerami są języki programowania. W miarę spadku kosztu sprzętu wzrósł względny koszt oprogramowania. Obok tego ciągły brak programistów spowodował bezwzględny wzrost kosztów oprogramowania. W wyniku tego głównym kosztem w cyklu życia systemu

jest oprogramowanie, a nie sprzęt. Poza kosztem, a także niewygodą, istnieje element zawodności: jest powszechne w odniesieniu do programów zarówno systemowych, jak i użytkowych, że ciągle, po latach wykorzystywania, są ujawniane nowe błędy.

Odpowiedzią badaczy i przemysłu było opracowywanie coraz potężniejszych i złożonych języków programowania wysokiego poziomu. Języki wysokiego poziomu umożliwiają programiście bardziej zwięzłe wyrażanie algorytmów, przejmują część troski o szczegóły i często w naturalny sposób wspierają użycie programowania strukturalnego.

Niestety, rozwiązanie to spowodowało powstanie innego problemu nazwanego *luką semantyczną*. Chodzi o różnicę między operacjami wyrażonymi w języku wysokiego poziomu a realizowanymi w architekturze komputera. Do symptomów tej luki należą nieefektywność wykonywania, nadmierne rozmiary programów maszynowych oraz złożoność kompilatorów. Odpowiedzią projektantów było opracowanie architektur umożliwiających zmniejszenie tej luki, które oferowały rozbudowane listy rozkazów, dziesiątki trybów adresowania wdrożone sprzętowo i różne dyrektywy języków wysokiego poziomu. Przykładem tej ostatniej możliwości jest rozkaz maszynowy CASE w systemie VAX. Tak złożone listy rozkazów zaprojektowano w celu:

- ułatwienia zadania programistom tworzącym kompilatory;
- poprawienia efektywności wykonywania, ponieważ złożone sekwencje operacji mogą być wdrożone w postaci mikrorozkazu;
- wspierania nawet bardziej złożonych i wyrafinowanych języków wysokiego poziomu.

Równocześnie, przez lata prowadzono badania w celu określenia właściwości i wzorów wykonywania rozkazów maszynowych generowanych przez programy pisane w językach wysokiego poziomu. Wyniki tych badań zainspirowały niektórych badaczy do rozważenia całkiem nowego rozwiązania: mianowicie do opracowania architektury, która wspierałaby raczej prostsze niż bardziej złożone języki wysokiego poziomu.

Tak więc, aby móc podążyć tokiem rozumowania zwolenników architektury RISC, rozpoczniemy od krótkiego przeglądu właściwości wykonywania rozkazów. Interesującymi aspektami obliczania są:

- **Przeprowadzane operacje.** Określają one działania prowadzone przez procesor i jego współpracę z pamięcią.
- **Używane argumenty.** Rodzaje argumentów i częstość ich używania determinują organizację pamięci służącej do ich przechowywania oraz tryby adresowania.
- **Szeregowanie rozkazów.** Określa ono organizację sterowania i przetwarzania potokowego.

W pozostałej części tego podrozdziału podsumujemy wyniki badań nad programami w językach wysokiego poziomu. Wszystkie wyniki są oparte na pomiarach dynamicznych. Oznacza to, że pomiary były gromadzone poprzez wykonywanie programu i liczenie częstości pojawiania się pewnej właściwości. W przeciwieństwie do

tego pomiary statyczne polegają na obliczeniach prowadzonych na tekście źródłowym programu. Nie dają one użytecznej informacji na temat wydajności, ponieważ nie są ważone zależnie od częstości wykonywania każdej instrukcji.

Operacje

Przeprowadzono wiele badań w celu przeanalizowania zachowywania się programów w językach wysokiego poziomu. W tabeli 4.7 przedstawionej w rozdziale 4 zawarto główne wyniki niektórych badań. Badania te dotyczyły pewnego zbioru języków i zastosowań i wykazały dobrą zgodność. Dominowały instrukcje przypisania, co sugeruje ważność prostego przemieszczania danych. Wystąpiła też przewaga instrukcji warunkowych (IF, LOOP). Instrukcje te są wdrażane w języku maszynowym za pomocą rozkazów porównywania i rozgałęzienia. Sugeruje to ważność mechanizmu sterowania szeregowaniem rozkazów.

Wyniki te są bardzo pouczające dla projektanta list rozkazów maszynowych, wskazują bowiem, jakie rodzaje instrukcji występują najczęściej i wobec tego powinny być optymalnie realizowane. Jednakże wyniki te nie ujawniają, które instrukcje zajmują najwięcej czasu przy wykonywaniu typowego programu. Przy danym skompilowanym programie w języku maszynowym chodzi o to, które instrukcje w języku źródłowym powodują wykonywanie większości rozkazów w języku maszynowym.

W celu zbadania tego podstawowego zjawiska skompilowano na maszynach VAX, PDP-11 i Motorola 68000 programy Pattersona [PATT82a] opisane w dodatku 4A, aby wyznaczyć średnią liczbę rozkazów maszynowych i odniesień do pamięci przypadającą na określony rodzaj instrukcji. Kolumny 2 i 3 w tabeli 13.2 ukazują względną częstość występowania różnych rozkazów w języku wysokiego poziomu w różnych programach; dane te zostały uzyskane drogą obserwacji ich występowania w wykonywanych programach, a nie na podstawie ich liczby w kodzie źródłowym. Są to więc dane statystyczne dotyczące dynamicznej częstości występowania rozkazów. W celu uzyskania danych zawartych w kolumnach 4 i 5 (ważone rozkazy maszynowe), każda wartość w drugiej i trzeciej kolumnie została pomnożona przez liczbę rozkazów maszynowych utworzonych przez kompilator. Wyniki te zostały następnie znormalizowane, dzięki czemu w kolumnach 4 i 5 mamy do czynienia ze względną częstością występowania, przy czym wagą jest liczba rozkazów maszynowych przypadająca na jedną instrukcję w języku wysokiego poziomu. Podobnie kolumny 6 i 7 zostały uzyskane przez pomnożenie częstości występowania poszczególnych rodzajów instrukcji przez względną liczbę odniesień do pamięci powodowaną przez te instrukcje. Dane zawarte w kolumnach od 4 do 7 stanowią zastępcze miary rzeczywistego czasu wykonywania różnych rodzajów instrukcji. Wyniki te wskazują, że najbardziej czasochłonnymi operacjami w typowych programach w językach wysokiego poziomu są wywołania i powroty procedur.

Znaczenie tabeli 13.2 powinno być dla czytelnika jasne. Pokazuje ona względne znaczenie różnych rodzajów instrukcji w języku wysokiego poziomu, gdy język ten jest skompilowany dla typowej, współczesnej architektury listy rozkazów. W przypadku odniesienia do innej architektury wyniki zapewne byłyby inne. Jednak

badania te doprowadziły do rezultatów, które są reprezentatywne dla współczesnych architektur komputerów o złożonej liście rozkazów (CISC). Mogą więc stanowić przewodnik dla tych, którzy szukają efektywniejszych dróg wspierania języków wysokiego poziomu.

Tabela 13.2. Ważona względna częstość operacji w językach wysokiego poziomu [PATT82a]

	Występowanie dynamiczne		Ważona według rozkazów maszynowych		Ważona według odniesień do pamięci	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	–	3%	–	–	–	–
W innych przypadkach	6%	1%	3%	1%	2%	1%

Argumenty

O wiele mniej prac poświęcono występowaniu różnych rodzajów argumentów, mimo ważności tego problemu. Występuje tu kilka znaczących aspektów.

Wspomniane już badania Pattersona [PATT82a] obejmowały również dynamiczną częstość występowania klas zmiennych (tabela 13.3). Wyniki, zgodne między programami w Pascalu i C, wykazały, że większość odniesień dotyczyła prostych zmiennych skalarnych. Ponad 80% tych skalarów to zmienne lokalne (w stosunku do procedury). Ponadto odniesienia do tablic i struktur wymagały uprzedniego odniesienia do ich indeksów lub wskaźników, które znów są na ogół lokalnymi skalarami. Występuje wobec tego przewaga odniesień do skalarów, a te z kolei są wysoce zlokalizowane.

Tabela 13.3. Dynamiczny udział procentowy argumentów

	Pascal	C	Średnio
Stałe całkowite	16%	23%	20%
Zmienne skalarne	58%	53%	55%
Tablice/struktury	26%	24%	25%

Badania Pattersona dotyczyły też dynamicznego zachowywania się programów w językach wysokiego poziomu, niezależnie od tła w postaci architektury. Jak stwierdzono poprzednio, aby głębiej przeanalizować zachowywanie się programów, konieczne jest oparcie się na rzeczywistych architekturach. W jednym z badań [LUND77] przeanalizowano dynamicznie rozkazy DEC-10 i stwierdzono, że przeciętnie każdy rozkaz odnosi się do 0,5 argumentu w pamięci i 1,4 w rejestrach. O podobnych wynikach poinformowano w [HUCK83] w odniesieniu do programów w językach C, Pascal i Fortran na S/370, PDP-11 i VAX. Oczywiście liczby te zależą silnie zarówno od architektury, jak i od kompilatora, jednak ilustrują częstość dostępu do argumentów.

Te ostatnie badania sugerują ważność takiej architektury, która jest podporządkowana szybkiemu dostępowi do argumentów, ponieważ ta właśnie operacja jest tak często wykonywana. Z badań Pattersona wynika, że pierwszym kandydatem do optymalizacji jest mechanizm przechowywania i osiągania lokalnych zmiennych skalarnych.

Wywołania procedur

Wiemy już, że wywołania procedur i powroty są ważnym aspektem programów w językach wysokiego poziomu. Udowodniono (tabela 13.2), że są one najbardziej czasochłonnymi operacjami w skompilowanych programach w językach wysokiego poziomu. Powinno więc być korzystne rozważenie sposobów efektywnego wdrażania tych operacji. Dwa aspekty są znaczące: liczba parametrów i zmiennych, które obejmuje dana procedura, oraz głębokość zagnieżdżenia.

W badaniach Tanenbauma [TANE78] zostało stwierdzone, że 98% dynamicznie wywoływanych procedur używa mniej niż 6 argumentów, zaś 92% z nich używa mniej niż 6 lokalnych zmiennych skalarnych. O podobnych wynikach doniósł zespół Berkeley RISC [KATE83], co widać w tabeli 13.4. Wyniki te wskazują, że liczba potrzebnych słów przypadająca na jedno wywołanie procedury nie jest wielka. Badania wspomniane wcześniej wykazały, że znaczną część odniesień do argumentów stanowią odniesienia do lokalnych zmiennych skalarnych. Wykazały one również, że odniesienia te są w rzeczywistości ograniczone do stosunkowo niewielu zmiennych.

Tabela 13.4. Argumenty procedur i lokalne zmienne skalarne

Udział wykonywanych wywołań podprogramów z:	Kompilator, interpreter i składopis	Małe programy nienumeryczne
> 3 argumentami	0÷7%	0÷5%
> 5 argumentami	0÷3%	0%
> 8 słowami argumentów i skalarów lokalnych	1÷20%	0÷6%
> 12 słowami argumentów i skalarów lokalnych	1÷6%	0÷3%

Ten sam zespół z Berkeley analizował również wzór wywołań i powrotów procedur w programach w językach wysokiego poziomu. Stwierdził on, że rzadkością są długie, nieprzerwane sekwencje wywołań procedur, po których następują odpowiednie sekwencje powrotów. Program pozostawał raczej ograniczony do wąskiego zakresu głębokości wywoływania procedur. Widać to na rys. 4.16, który był omówiony w rozdz. 4. Wyniki te stanowią potwierdzenie konkluzji, że odniesienia do argumentów są wysoce zlokalizowane.

Wnioski

Wiele zespołów zapoznawało się z takimi wynikami, jak przedstawione powyżej, i dochodziło do wniosku, że dążenie do dostosowania architektury listy rozkazów do języka wysokiego poziomu nie jest najbardziej efektywną strategią projektowania.

Najkorzystniejszym wsparciem języka wysokiego poziomu jest optymalizacja najbardziej czasochłonnych składników typowych programów w tych językach.

Uogólniając wyniki prac wielu badaczy, możemy stwierdzić, że architektury RISC można w całości scharakteryzować za pomocą trzech elementów. Po pierwsze, w architekturze RISC występuje duża liczba rejestrów. Ma to na celu zoptymalizowanie odnoszenia się do argumentów. Z przedstawionych wyżej wyników badań można wywnioskować, że na jedną instrukcję w języku wysokiego poziomu przypada kilka odniesień i że występuje znaczny udział instrukcji przenoszenia (przypisania). W połączeniu z lokalnością i z dominacją odniesień skalarnych sugeruje to, że wydajność może być poprawiona przez zredukowanie odniesień do pamięci kosztem większej liczby odniesień do rejestrów. Ze względu na lokalność tych odniesień rozszerzony zestaw rejestrów wydaje się być praktyczny.

Po drugie, w architekturze RISC należy poświęcić wiele uwagi projektowaniu potoków rozkazów. Ze względu na wysoki udział rozkazów rozgałęzienia warunkowego oraz wywołania procedury, prosty potok rozkazów nie będzie efektywny. Uwidacznia się to poprzez wysoki udział rozkazów, które są wstępnie pobierane i nigdy nie wykonywane.

Po trzecie wreszcie, wskazana jest tu uproszczona (zredukowana) lista rozkazów. Punkt ten nie jest tak oczywisty, jak pozostałe, jednak stanie się jaśniejszy podczas dalszej dyskusji.

13.2. Użycie dużej tablicy rejestrów

Z rezultatów badań podsumowanych w podrozdz. 13.1 wynika, że pożądany jest szybki dostęp do argumentów. Widzieliśmy, że w programach w językach wysokiego poziomu występuje duży odsetek instrukcji przypisania i że wiele z nich ma prostą postać A←B. Na jedną instrukcję w języku wysokiego poziomu przypada też znaczna liczba odniesień do argumentów. Jeśli połączymy te stwierdzenia z faktem, że większość z tych odniesień dotyczy lokalnych skalarów, wyniknie stąd sugestia, aby polegać na przechowywaniu w rejestrach.

Przechowywanie w rejestrach jest wskazane, ponieważ są one najszybszymi przyrządami do przechowywania, szybszymi niż pamięć główna i pamięć podręczna. Tablica rejestrów jest fizycznie mała. Znajduje się na ogół w tym samym mikroukładzie co ALU i jednostka sterująca oraz wymaga znacznie krótszych adresów niż pamięci główna i podręczna. Potrzebna jest więc strategia, która pozwoli na to, żeby najczęściej powoływane argumenty były przechowywane w rejestrach i żeby operacje rejestr-pamięć były zminimalizowane.

Możliwe są dwa podstawowe rozwiązania: jedno oparte na oprogramowaniu, drugie na sprzęcie. W rozwiązaniu programowym polega się na maksymalizacji użycia rejestrów przez kompilator. Kompilator może dążyć do przydzielenia rejestrów tym zmiennym, które w określonym czasie będą najczęściej używane. Rozwiązanie to wymaga użycia wyrafinowanych algorytmów analizy programów. Podejście sprzę-

towe polega po prostu na użyciu większej liczby rejestrów, dzięki czemu więcej zmiennych można przetrzymywać w rejestrach przez dłuższe okresy.

W tym podrozdziale przedyskutujemy rozwiązanie sprzętowe. Zostało ono zapoczątkowane przez grupę RISC z Berkeley [PATT82a] i było zastosowane w pierwszym komercyjnym wyrobie o architekturze RISC, jakim był procesor Pyramid [RAGA83]. Jest ono obecnie stosowane w popularnej architekturze SPARC.

Okna rejestrów

Użycie dużego zestawu rejestrów przede wszystkim powinno ograniczyć potrzebę sięgania do pamięci. Projektowanie powinno polegać na zorganizowaniu rejestrów w taki sposób, żeby cel ten został osiągnięty.

Ponieważ większość odniesień do pamięci dotyczy lokalnych skalarów, oczywistym rozwiązaniem jest przechowywanie ich w rejestrach, z jednoczesnym ewentualnym pozostawieniem kilku rejestrów na zmienne globalne. Problem polega na tym, że pojęcie *lokalności* zmienia się wraz z każdym wywołaniem procedury i powrotem, a więc z operacjami, które występują często. Przy każdym wywołaniu zmienne lokalne muszą być zachowane poprzez przeniesienie z rejestrów do pamięci, dzięki czemu rejestry mogą być użyte ponownie przez wywołany program. Ponadto muszą być przekazane parametry. Przy powrocie zmienne programu macierzystego muszą być odtworzone (załadowane z powrotem do rejestrów), a wyniki muszą być przekazane z powrotem do programu macierzystego.

Rozwiązanie jest oparte na dwóch innych rezultatach badań, które omówiliśmy w podrozdz. 13.1. Po pierwsze, typowa procedura wykorzystuje tylko kilka przekazanych parametrów i zmiennych lokalnych (tabela 13.4). Po drugie, głębokość aktywacji procedury ulega wahaniom w stosunkowo wąskim zakresie (rys. 4.16). W celu wykorzystania tych własności używa się wielu małych zestawów rejestrów, z których każdy jest przypisany do innej procedury. Wywołanie procedury powoduje automatyczne przełączenie procesora do użycia innego okna rejestrów o ustalonym rozmiarze, zamiast zapisywania zawartości rejestrów w pamięci. Okna sąsiednich procedur nakładają się, co umożliwia przekazywanie parametrów.

Koncepcja ta jest zilustrowana na rys. 13.1. W dowolnej chwili widzialne i adresowalne jest tylko jedno okno rejestrów, tak jak gdyby istniał tylko jeden zestaw rejestrów (o adresach np. od 0 do $N - 1$). Okno jest podzielone na trzy obszary o stałych rozmiarach. W rejestrach parametrów przechowuje się parametry przekazane przez procedurę, która spowodowała wywołanie bieżącej procedury, oraz wyniki, które mają być przekazane z powrotem. Rejestry lokalne są używane do przechowywania zmiennych lokalnych przypisanych przez kompilator. Rejestry tymczasowe służą do wymiany parametrów i wyników z następnym niższym poziomem (z procedurą wywołaną przez procedurę bieżącą). Rejestry tymczasowe jednego poziomu są fizycznie tymi samymi rejestrami co rejestry parametrów następnego niższego poziomu. To nakładanie się umożliwia przekazywanie parametrów bez rzeczywistego przenoszenia danych.



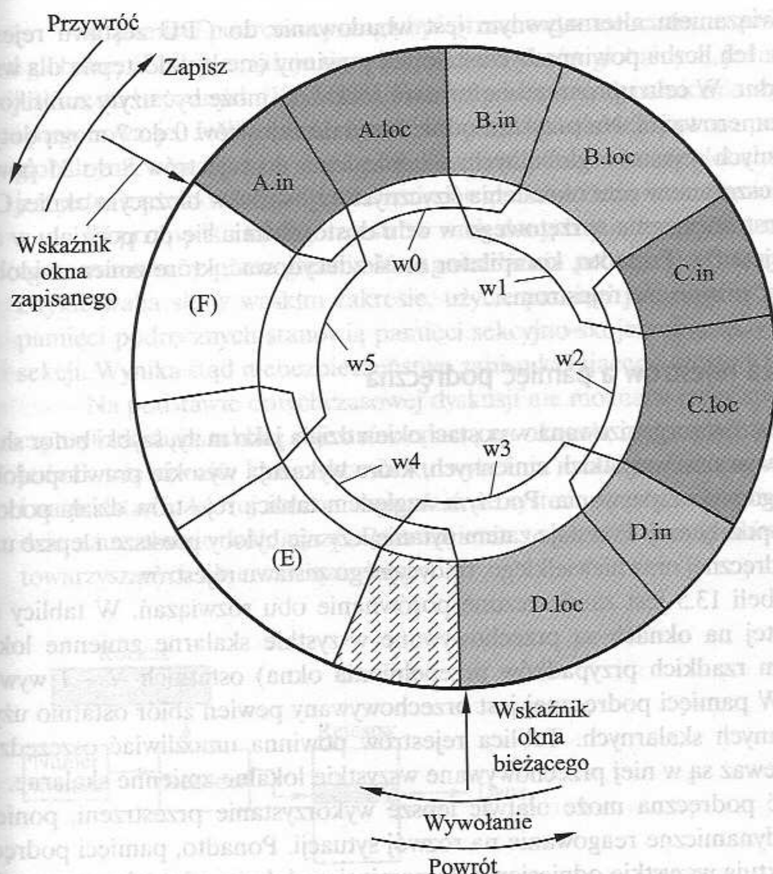
Rysunek 13.1. Nakładające się okna rejestrów

Wydawałoby się, że w celu sprostania dowolnym możliwym układom wywołań i powrotów liczba okien rejestrów powinna być nieograniczona. W rzeczywistości okna rejestrów mogą być używane do przechowywania argumentów kilku ostatnio wywoływanych procedur. Argumenty starszych procedur muszą być zapisywane w pamięci oraz odtwarzane później, gdy maleje głębokość zagnieżdżenia. Wobec tego rzeczywista organizacja tablicy rejestrów ma charakter cyklicznego bufora złożonego z nakładających się częściowo okien. Dwoma godnymi uwagi przykładami takiego podejścia są: architektura SPARC firmy Sun opisana w podrozdz. 13.7 oraz architektura IA-64 zastosowana w procesorze Itanium firmy Intel, opisanym w rozdz. 15.

Organizację tę widać na rys. 13.2, na którym jest pokazany cykliczny bufor złożony z 6 okien. Bufor jest napełniony do poziomu 4 (A wywołała B; B wywołała C; C wywołała D), przy czym aktywna jest procedura D. Wskaźnik bieżącego okna (CWP) wskazuje okno aktualnie aktywnej procedury. Odniesienia do rejestrów ze strony rozkazów maszynowych są przesuwane za pomocą tego wskaźnika w celu określenia rzeczywistego rejestru fizycznego. Wskaźnik zapisanego okna (SWP) określa okno, którego zawartość została ostatnio przekazana do pamięci. Jeśli procedura D wywoła teraz procedurę E, to argumenty procedury E są już umieszczone w rejestrach tymczasowych D (nakładanie się w3 i w2), a stan CWP jest zwiększany o jedno okno.

Jeśli następnie procedura E wywoła procedurę F, to wywołanie to nie może być zrealizowane przy aktualnym stanie bufora. Jest tak, ponieważ okno F nakłada się z oknem A. Procedura F, przygotowując się do wywołania, rozpoczęłaby ładowanie swoich rejestrów tymczasowych, wymieniłaby tym samym zawartość rejestrów parametrów A (A.in). Wobec tego, gdy stan CWP jest zwiększany (modulo 6) i staje się równy SWP, następuje przerwanie i zapisywana jest zawartość okna A. Zapisane muszą być tylko pierwsze dwie części (A.in i A.loc). Następnie zwiększany jest stan SWP, a procedura F postępuje. Podobne przerwanie może nastąpić przy powrocie. Jeśli na przykład po wzbudzeniu procedury F procedura B wraca do A, stan CWP jest zmniejszany i staje się równy SWP. Powoduje to przerwanie, którego wynikiem jest odtworzenie okna procedury A.

Na podstawie powyższego opisu można zauważyć, że w tablicy rejestrów o N oknach można przechowywać tylko $N-1$ wywołań procedur. Wartość N nie musi być duża. Jak wspomnieliśmy w dodatku 4A, w jednym z badań [TAMI83] stwierdzono,



Rysunek 13.2. Cykliczno-buforowa organizacja nakładających się okien

że w przypadku 8 okien zapisywanie lub odtwarzanie jest wymagane tylko dla 1% wywołań i powrotów. W komputerach RISC Berkeley użyto 8 okien po 16 rejestrów każde. W komputerze Pyramid zastosowano 16 okien po 32 rejestry każde.

Zmienne globalne

Opisany powyżej układ okien stanowi efektywną organizację służącą do przechowywania zmiennych skalarnych w rejestrach lokalnych. Układ ten nie odpowiada jednak potrzebom przechowywania zmiennych globalnych wykorzystywanych przez więcej niż jedną procedurę. Dwie opcje nasuwają się same. Po pierwsze, zmienne zadeklarowane w języku wysokiego poziomu jako globalne mogą być przypisane przez kompilator lokacjom w pamięci i wszystkie rozkazy maszynowe odnoszące się do tych zmiennych będą używały argumentów zawartych w pamięci. Jest to proste zarówno pod względem sprzętu, jak i oprogramowania (kompilatora). Jednak w przypadku często wywoływanych zmiennych globalnych układ ten jest nieefektywny.

Rozwiązaniem alternatywnym jest wbudowanie do CPU zestawu rejestrów globalnych. Ich liczba powinna być ustalona i powinny one być dostępne dla wszystkich procedur. W celu uproszczenia formatu rozkazów może być użyty zunifikowany schemat numerowania. Na przykład odniesienia do rejestrów 0 do 7 mogą dotyczyć jednoznacznych rejestrów globalnych, a odniesienia do rejestrów 8 do 31 powinny być przemieszczane w celu określenia fizycznych rejestrów w bieżącym oknie. Oznacza to wzrost obciążenia sprzętowego w celu dostosowania się do podziału w adresowaniu rejestrów. Ponadto, kompilator musi zdecydować, które zmienne globalne powinny być przypisane rejestrom.

Duża tablica rejestrów a pamięć podręczna

Tablica rejestrów zorganizowana w postaci okien działa jako mały, szybki bufor służący do przechowywania wszystkich zmiennych, które wykazują wysokie prawdopodobieństwo częstego wykorzystywania. Pod tym względem tablica rejestrów działa podobnie do pamięci podręcznej. Powstaje zatem pytanie, czy nie byłoby prostsze i lepsze użycie pamięci podręcznej oraz niewielkiego, tradycyjnego zestawu rejestrów.

W tabeli 13.5 jest zamieszczone porównanie obu rozwiązań. W tablicy rejestrów opartej na oknach są przechowywane wszystkie skalarne zmienne lokalne (z wyjątkiem rzadkich przypadków przepełnienia okna) ostatnich $N - 1$ wywołań procedur. W pamięci podręcznej jest przechowywany pewien zbiór ostatnio używanych zmiennych skalarnych. Tablica rejestrów powinna umożliwiać oszczędzanie czasu, ponieważ są w niej przechowywane wszystkie lokalne zmienne skalarne. Jednak pamięć podręczna może ułatwić lepsze wykorzystanie przestrzeni, ponieważ umożliwia dynamiczne reagowanie na rozwój sytuacji. Ponadto, pamięci podręczne na ogół traktują wszystkie odniesienia do pamięci podobnie, włączając w to rozkazy i inne rodzaje danych. Zatem oszczędności w tych innych obszarach są możliwe, jeśli chodzi o pamięć podręczną, a nie występują w przypadku tablicy rejestrów.

Tablica rejestrów może nieefektywnie wykorzystywać przestrzeń, ponieważ nie wszystkie procedury będą potrzebowały pełnej przestrzeni przypisanych im okien. Z drugiej strony pamięć podręczna charakteryzuje się nieefektywnością innego rodzaju: dane są wczytywane do pamięci podręcznej blokami. Podczas gdy tablica rejestrów zawiera tylko używane zmienne, do pamięci podręcznej są wczytywane bloki danych, z których wiele nie będzie używanych.

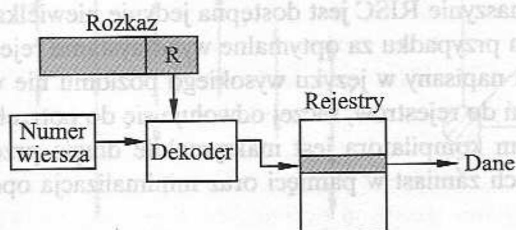
Tabela 13.5. Właściwości organizacji opartych na dużych tablicach rejestrów oraz na pamięciach podręcznych

Duża tablica rejestrów	Pamięć podręczna
Wszystkie skalary lokalne	Ostatnio używane skalary lokalne
Pojedyncze zmienne	Bloki pamięci
Zmienne globalne przypisane przez kompilator	Ostatnio używane zmienne globalne
Zachowywanie/odtwarzanie oparte na głębokości zagnieżdżenia procedury	Zachowywanie/odtwarzanie oparte na algorytmie wymiany pamięci podręcznej
Adresowanie rejestrów	Adresowanie pamięci

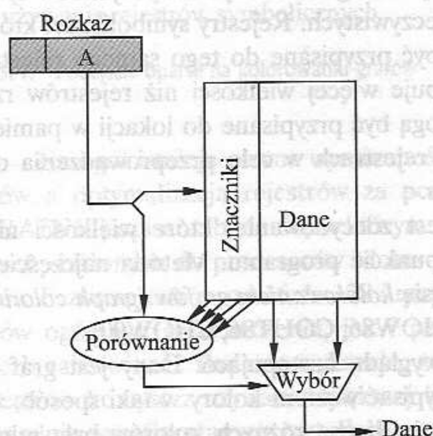
W pamięci podręcznej mogą być przechowywane zarówno zmienne lokalne, jak i globalne. Zwykle znajduje się w niej wiele skalarów globalnych, jednak niewiele z nich jest często używanych [KATE83]. Pamięć podręczna dynamicznie wykryje te zmienne i przechowa je. Jeśli tablica rejestrów oparta na oknach jest uzupełniona rejestrami globalnymi, ona również może przechowywać pewne skalary globalne. Kompilatorowi jest jednak trudno określić, które zmienne globalne będą intensywnie używane.

W przypadku tablicy rejestrów ruch danych między rejestrami a pamięcią jest zdeterminowany przez głębokość zagnieżdżenia procedury. Ponieważ głębokość ta zwykle waha się w wąskim zakresie, użycie pamięci jest niezbyt częste. Większość pamięci podręcznych stanowią pamięci sekcyjno-skojarzeniowe o małym rozmiarze sekcji. Wynika stąd niebezpieczeństwo zapisu kasującego często używane zmienne.

Na podstawie dotychczasowej dyskusji nie można w łatwy sposób zdecydować, czy wybrać dużą tablicę rejestrów opartą na oknach, czy pamięć podręczną. Istnieje jednak właściwość, która skłania ku wybraniu rozwiązania opartego na rejestrach i na podstawie której można stwierdzić, że system oparty na pamięci podręcznej będzie zauważalnie wolniejszy. Rozróżnienie to wynika z nakładów na adresowanie towarzyszących obu rozwiązaniom.



(a) Tablica rejestrów oparta na oknach



(b) Pamięć podręczna

Rysunek 13.3. Odniesienie do skalara lokalnego

Różnicę tę widać na rys. 13.3. W celu odniesienia do lokalnego skalara w tablicy rejestrów opartej na oknach jest używany „wirtualny" numer rejestru oraz numer okna. Mogą one być przepuszczone przez stosunkowo prosty dekoder w celu wybrania jednego z rejestrów fizycznych. W celu odniesienia do lokacji w pamięci podręcznej musi być wygenerowany adres pamięci o pełnej szerokości. Złożoność tej operacji zależy od trybu adresowania. W sekcyjno-skojarzeniowej pamięci podręcznej część adresu jest używana do odczytania liczby słów i znaczników równej rozmiarowi sekcji. Inna część adresu jest porównywana ze wskaźnikami i wybierane jest jedno z odczytanych słów. Powinno być jasne, że jeśli nawet pamięć podręczna będzie tak samo szybka jak tablica rejestrów, to czas dostępu będzie znacznie dłuższy. Wobec tego, pod względem wydajności, tablica rejestrów oparta na oknach jest lepsza do przechowywania lokalnych skalarów. Dalsza poprawa wydajności może być uzyskana przez dodanie pamięci podręcznej przeznaczonej wyłącznie do przechowywania rozkazów.

13.3. Optymalizacja rejestrów za pomocą kompilatora

Załóżmy teraz, że w docelowej maszynie RISC jest dostępna jedynie niewielka liczba rejestrów (np. 16÷32). W tym przypadku za optymalne wykorzystanie rejestrów odpowiada kompilator. Program napisany w języku wysokiego poziomu nie wykazuje oczywiście jawnych odniesień do rejestrów, raczej odwołuje się do potrzebnych wielkości symbolicznie. Zadaniem kompilatora jest maksymalnie długie przetrzymywanie argumentów w rejestrach zamiast w pamięci oraz minimalizacja operacji ładowania i zapisu.

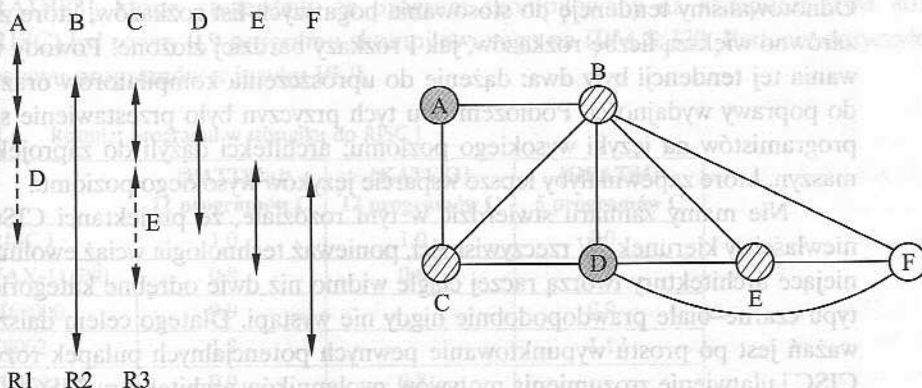
Na ogół stosuje się następujące rozwiązanie. Każda wielkość programu mająca pozostawać w rejestrach jest przypisana do rejestru symbolicznego lub wirtualnego. Następnie kompilator odwzorowuje nieograniczoną liczbę rejestrów symbolicznych na ustalonej liczbie rejestrów rzeczywistych. Rejestry symboliczne, których zastosowania nie nakładają się, mogą być przypisane do tego samego rejestru. Jeśli w określonej części programu występuje więcej wielkości niż rejestrów rzeczywistych, to niektóre z tych wielkości mogą być przypisane do lokacji w pamięci. Aby tymczasowo umieścić te wielkości w rejestrach w celu przeprowadzenia operacji, używa się rozkazów ładowania i zapisu.

Istotą zadania optymalizacji jest zdecydowanie, które wielkości mają być przypisane rejestrom w określonym punkcie programu. Metoda najczęściej stosowana w kompilatorach RISC nazywa się *kolorowaniem grafów* (*graph coloring*) i zapożyczono ją z topologii [CHAI82, CHOW86, COUT86, CHOW90].

Problem kolorowania grafów wygląda następująco. Dany jest graf złożony z węzłów i linii łączących. Należy przypisać węzłom kolory w taki sposób, żeby sąsiednie węzły miały różne kolory i żeby liczba różnych kolorów była minimalna. Problem ten został zaadaptowany do tworzenia kompilatorów. Po pierwsze, analizuje się program w celu zbudowania grafu (wykresu) interferencji rejestrów. Węzłami grafu są rejestry symboliczne. Jeśli dwa rejestry symboliczne są aktywne w tym sa-

mym fragmencie programu, łączy się je linią odzwierciedlającą interferencję. Następnie próbuje się pokolorować graf za pomocą *n* kolorów, gdzie *n* jest liczbą rejestrów. Węzły o takim samym kolorze mogą być przypisane temu samemu rejestrowi. Jeśli proces ten nie w pełni się udaje, to węzły, które nie mogą być pokolorowane, muszą być umieszczone w pamięci. Gdy będą one potrzebne, miejsce dla nich jest tworzone za pomocą rozkazów ładowania i zapisu.

Na rysunku 13.4 widać prosty przykład takiego procesu. Załóżmy, że program o 6 rejestrach symbolicznych ma być skompilowany do 3 rejestrów rzeczywistych. Na rysunku 13.4a jest zilustrowana sekwencja czasowa aktywacji każdego rejestru symbolicznego, a w części b – graf interferencji rejestrów (cieniowanie i kreskowanie zastępują kolory). Pokazano możliwość pokolorowania za pomocą trzech kolorów. Jeden z rejestrów symbolicznych, F, pozostał niepokolorowany i musi być używany za pomocą operacji ładowania i zapisu.



(a) Sekwencja czasowa aktywnego używania rejestrów symbolicznych

(b) Graf interferencji rejestrów

Rysunek 13.4. Podejście oparte na kolorowaniu grafów

Na ogół istnieje pewna wymienność między używaniem dużego zestawu rejestrów a optymalizacją rejestrów za pomocą kompilatora. Na przykład w pracy [BRAD91a] opisano badanie, w którym modelowano architekturę RISC o własnościach zbliżonych do procesorów Motorola 88000 i MIPS R2000. W badaniu zmieniano liczbę rejestrów od 16 do 128 i rozpatrywano zarówno używanie ich jako rejestrów ogólnego przeznaczenia, jak i podzielenie rejestrów na zmiennopozycyjne i liczb całkowitych. Badanie wykazało, że nawet przy zrównoważonej optymalizacji rejestrów korzyść z używania większej liczby rejestrów niż 64 jest niewielka. Przy rozsądnie wyrafinowanych metodach optymalizacji liczby rejestrów, już powyżej 32 rejestrów występuje marginalna poprawa wydajności. Stwierdzono również, że przy małej liczbie rejestrów (np. 16) maszyna o wspólnych rejestrach funkcjonuje szybciej niż maszyna o podzielonych rejestrach. Podobne wnioski można wyciągnąć

z pracy [HUGU91], w której są opisane badania poświęcone bardziej optymalizacji użycia małej liczby rejestrów, niż porównywaniu użycia dużych zestawów rejestrów w połączeniu z optymalizacją.

13.4. Architektura o zredukowanej liście rozkazów

W tym podrozdziale rozpatrzymy niektóre własności ogólne i sposoby stosowania architektury o zredukowanej liście rozkazów. Przykłady przedstawimy w dalszej części rozdziału. Rozpoczniemy od przedyskutowania powodów stosowania współczesnych architektur o złożonej liście rozkazów.

Dlaczego CISC

Odnotowaliśmy tendencję do stosowania bogatszych list rozkazów, które zawierają zarówno większą liczbę rozkazów, jak i rozkazy bardziej złożone. Powody występowania tej tendencji były dwa: dążenie do uproszczenia kompilatorów oraz dążenie do poprawy wydajności. Podłożem obu tych przyczyn było przestawienie się części programistów na języki wysokiego poziomu; architekci dążyli do zaprojektowania maszyn, które zapewniałyby lepsze wsparcie języków wysokiego poziomu.

Nie mamy zamiaru stwierdzić w tym rozdziale, że projektanci CISC obrali niewłaściwy kierunek. W rzeczywistości, ponieważ technologia wciąż ewoluuje, a istniejące architektury tworzą raczej ciągłe widmo niż dwie odrębne kategorie, ocena typu czarne–białe prawdopodobnie nigdy nie wystąpi. Dlatego celem dalszych rozważań jest po prostu wypunktowanie pewnych potencjalnych pułapek rozwiązania CISC i ułatwienie zrozumienia motywów zwolenników architektury RISC.

Pierwsza spośród wymienionych przyczyn, to znaczy uproszczenie kompilatorów, wygląda na oczywistą. Zadaniem twórcy kompilatorów jest generowanie ciągu rozkazów maszynowych dla każdej instrukcji w języku wysokiego poziomu. Jeśli istnieją rozkazy maszynowe, które zastępują instrukcje w języku wysokiego poziomu, to zadanie to jest uproszczone. Takie rozumowanie zostało przedyskutowane przez badaczy RISC [HENN82, RADI83, PATT82b]. Stwierdzili oni, że złożone rozkazy maszynowe są często trudne do wykorzystania, ponieważ kompilator musi odnaleźć te przypadki, które dokładnie pasują do konstrukcji. Zadanie optymalizacji generowanego kodu w celu minimalizacji jego rozmiaru, zredukowania liczby wykonań rozkazów i spotęgowania potokowania jest o wiele trudniejsze w przypadku złożonej listy rozkazów. Dowodem na to są cytowane wcześniej badania wykazujące, że większość rozkazów w skompilowanych programach stanowią rozkazy stosunkowo proste.

Inną ważną, cytowaną już przyczyną jest oczekiwanie, że CISC pozwoli uzyskiwać mniejsze i szybsze programy. Przeanalizujmy oba aspekty stwierdzenia: że programy będą mniejsze i że będą szybciej realizowane.

Mniejsze programy mają dwie zalety. Po pierwsze, występują oszczędności pamięci. Wobec taniości współczesnych pamięci ta potencjalna zaleta nie jest już tak atrakcyjna. Ważniejsze jest to, że mniejsze programy powinny umożliwić popra-

wienie wydajności i to na dwa sposoby. Po pierwsze, mniej rozkazów oznacza mniej bajtów do pobrania. Po drugie zaś, w środowisku stronicowanym mniejsze programy zajmują mniej stron, co powoduje zmniejszenie liczby błędów stron.

Problemem występującym w tej linii rozumowania jest to, że nie jest pewne, czy program CISC będzie mniejszy niż odpowiedni program RISC. W wielu przypadkach program CISC wyrażony w symbolicznym języku maszynowym może być *krótszy* (to znaczy może zawierać mniej rozkazów), jednak liczba zajmowanych bitów pamięci może nie być zauważalnie *mniejsza*. W tabeli 13.6 są pokazane wyniki trzech badań, w których porównywano rozmiary skompilowanych programów w języku C dla różnych maszyn łącznie z RISC I o zredukowanej liście rozkazów. Zauważmy, że CISC przynosi nieznaczne lub nie przynosi żadnych oszczędności w porównaniu z RISC. Jest również interesujące, że komputer VAX dysponujący o wiele bardziej złożoną listą rozkazów niż PDP-11 umożliwia osiągnięcie niewielkich oszczędności w porównaniu z tym ostatnim. Wyniki te zostały potwierdzone przez badaczy IBM [RADI83], którzy stwierdzili, że program skompilowany na komputer IBM 801 (RISC) był równy 0,9 programu skompilowanego na IBM S/370. Badanie dotyczyło zestawu programów w języku PL/I.

Tabela 13.6. Rozmiar programu w stosunku do RISC I

	[PATT82a] 11 programów C	**[KATE83] 12 programów C**	**[HEAT84] 5 programów C**
RISC I	1,0	1,0	1,0
VAX-11/780	0,8	0,67	
M68000	0,9		0,9
Z8002	1,2		1,12
PDP-11/70	0,9	0,71	

Istnieje kilka przyczyn tych raczej zaskakujących wyników. Zauważyliśmy już, że kompilatory dla CISC wykazują tendencję do faworyzowania prostszych rozkazów, więc zwięzłość złożonych rozkazów rzadko odgrywa rolę. Ponadto, ponieważ w CISC występuje więcej rozkazów, wymagane są dłuższe kody operacji, co powoduje wydłużenie rozkazów. Wreszcie maszyny RISC wykazują tendencję do akcentowania raczej odniesień do rejestrów niż do pamięci, a te pierwsze wymagają mniej bitów. Przykład tego ostatniego zjawiska omawiamy poniżej.

Tak więc oczekiwanie, że CISC umożliwi powstanie mniejszych programów wraz z towarzyszącymi im zaletami, może nie być spełnione. Drugim czynnikiem motywującym wzrastającą złożoność list rozkazów było przyspieszenie wykonywania rozkazów. Oczekiwanie, że złożona operacja w języku wysokiego poziomu zostanie wykonana szybciej jako pojedynczy rozkaz maszynowy niż jako ciąg bardziej prymitywnych rozkazów, wygląda na uzasadnione. Jednak ze względu na tendencję do stosowania prostszych rozkazów może to nie być prawdą. Jednostka sterująca musi być bardziej skomplikowana i (lub) przechowywanie mikroprogramów sterowania musi zajmować większą pojemność, co wynika z bogatszej listy rozkazów. Oba te czynniki wydłużają czas wykonywania prostych rozkazów.

8	16	16	16
Dodaj	B	C	A

Pamięć-pamięć
I = 56, D = 96, M = 152

8	4		16
Ładuj	rB		B
Ładuj	rC		C
Dodaj	rA	rB	rC
Zapisz	rA		A

Rejestr-pamięć
I = 104, D = 96, M = 200

(a) **A ← B + C**

8	16	16	16
Dodaj	B	C	A
Dodaj	A	C	B
Odejmij	B	D	D

Pamięć-pamięć
I = 168, D = 288, M = 456

8	4	4	4
Dodaj	rA	rB	rC
Dodaj	rB	rA	rC
Odejmij	rD	rD	rB

Rejestr-pamięć
I = 60, D = 0, M = 60

(b) **A ← B + C; B ← A + C; D ← D – B**

I – rozmiar wykonywanych rozkazów
D – rozmiar przetwarzanych danych
M = I + D = całkowity ruch do (i z) pamięci

Rysunek 13.5. Porównanie rozwiązań rejestr-rejestr oraz pamięć-pamięć

W rzeczywistości niektórzy badacze stwierdzili, że przyspieszenie wykonywania złożonych funkcji wynika nie tyle z potęgi złożonych rozkazów maszynowych, co z ich przechowywania w szybkiej pamięci sterującej [RADI83]. W rezultacie pamięć sterująca działa jako podręczna pamięć rozkazów. Wobec tego projektant sprzętu powinien spróbować określić, które podprogramy lub funkcje będą używane najczęściej, i wprowadzić je do pamięci sterującej przez wdrożenie ich w postaci mikrokodu. Wyniki okazały się mniej niż zachęcające. W związku z tym w systemach S/390 rozkazy takie, jak Translate and Extended-Precision-Floating-Point-Divide (translacja i dzielenie zmiennopozycyjne o zwiększonej dokładności) rezydują w szybkiej pamięci sterującej, podczas gdy sekwencje dotyczące tworzenia wywołań procedur lub inicjowania programu obsługi przerwań znajdują się w wolniejszej pamięci głównej.

Nie jest więc oczywiste, że tendencja do zwiększania złożoności list rozkazów jest właściwa. Skłoniło to wiele grup projektantów do wybrania przeciwstawnej drogi.

Własności architektur o zredukowanej liście rozkazów

Chociaż było wiele różnych podejść do architektury o zredukowanej liście rozkazów, pewne własności pozostają wspólne dla wszystkich:

- Jeden rozkaz na cykl.
- Operacje „z rejestru do rejestru".

- Proste tryby adresowania.
- Proste formaty rozkazów.

Przeanalizujemy teraz pokrótce te własności. Konkretne przykłady zostaną przedstawione w dalszym ciągu tego rozdziału.

Pierwszą z wymienionych własności jest to, że **na jeden cykl maszynowy przypada jeden rozkaz maszynowy**. *Cykl maszynowy* jest definiowany jako czas wymagany do pobrania 2 argumentów z rejestrów, przeprowadzenia operacji ALU i zapisania wyniku w rejestrze. Rozkazy RISC nie powinny więc być bardziej skomplikowane i wolniej wykonywane niż mikrorozkazy w maszynach CISC (co zostanie przeanalizowane w części 4). Przy prostych, 1-cyklowych rozkazach nie występuje potrzeba mikrokodu; rozkazy maszynowe mogą być implementowane sprzętowo. Takie rozkazy powinny być wykonywane szybciej niż porównywalne rozkazy w innych maszynach, ponieważ podczas wykonywania rozkazów nie ma potrzeby sięgania do mikroprogramu w pamięci sterującej.

Drugą własnością jest to, że większość operacji powinna mieć charakter **rejestr-rejestr**, jedynie w połączeniu z prostymi operacjami dostępu do pamięci (LOAD i STORE). Własność ta powoduje uproszczenie listy rozkazów i wobec tego jednostki sterującej. Lista rozkazów RISC może na przykład zawierać tylko jeden lub dwa rozkazy ADD (np. dodaj całkowite, dodaj z przeniesieniem); VAX ma 25 różnych rozkazów ADD. Inną korzyścią jest to, że taka architektura skłania do optymalizacji rejestrów, dzięki czemu często wywoływane argumenty pozostają w szybkiej pamięci.

Nacisk na operacje rejestr-rejestr jest charakterystyczny dla projektów RISC. Inne współczesne maszyny dysponują takimi rozkazami, jednak obejmują także operacje pamięć-pamięć oraz mieszane operacje rejestr-pamięć. Próby porównania tych rozwiązań podejmowano w latach siedemdziesiątych, jeszcze przed pojawieniem się architektur RISC. Na rysunku 13.5a jest pokazane przyjęte podejście. Hipotetyczne architektury oceniano pod kątem rozmiaru programów i liczby bitów zapisywanych i pobieranych z pamięci. Uzyskane wyniki doprowadziły jednego z badaczy do zasugerowania, że przyszłe architektury w ogóle nie powinny zawierać rejestrów [MYER78]. Można się zastanawiać, co sądziłby on dzisiaj, gdy firma Pyramid wprowadziła na rynek maszynę RISC zawierającą ni mniej, ni więcej, tylko 528 rejestrów!

W badaniach tych umknęło rozpoznanie częstego dostępu do małej liczby lokalnych skalarów oraz to, że dzięki dużemu bankowi rejestrów lub optymalizacji za pomocą kompilatorów większość argumentów może być przez długi czas trzymana w rejestrach. Dlatego też rys. 13.5b może stanowić uczciwsze porównanie.

Trzecią własnością jest użycie **prostych trybów adresowania**. W prawie wszystkich rozkazach RISC używa się prostego adresowania rejestrowego. Może też być włączonych kilka dodatkowych trybów, takich jak tryb oparty na przesunięciach lub względny wobec licznika rozkazów. Inne, bardziej złożone tryby mogą być składane programowo z prostszych. I znów własność ta umożliwia uproszczenie listy rozkazów i jednostki sterującej.

Ostatnią ze wspólnych własności jest używanie **prostych formatów rozkazu**. Na ogół stosuje się jeden lub kilka formatów rozkazu. Długość rozkazów jest stała i wyrównana z granicami słów. Położenia pól, w szczególności kodu systemu operacyjnego, są ustalone. Własność ta ma wiele zalet. Przy ustalonych polach, dekodowanie kodu operacji i dostęp do argumentów w rejestrach mogą być dokonywane jednocześnie. Uproszczone formaty umożliwiają uproszczenie jednostki sterującej. Pobieranie rozkazów jest zoptymalizowane, ponieważ pobierane są jednostki o długości słowa. Oznacza to także, że pojedyncze rozkazy nie wykraczają poza granice stron.

Własności te potraktowane łącznie mogą służyć do określenia potencjalnych korzyści z zastosowania rozwiązania RISC. Korzyści te można podzielić na dwie główne kategorie: związane z wydajnością i związane z wdrażaniem układów VLSI.

W odniesieniu do wydajności można przedstawić następujące „poszlaki". Po pierwsze, mogą być opracowane bardziej efektywne kompilatory optymalizujące. Przy bardziej prymitywnych rozkazach jest więcej sposobności do zmniejszania liczby pętli, zreorganizowania programów pod kątem efektywności, maksymalizacji wykorzystania rejestrów i tak dalej. Jest nawet możliwe obliczanie części złożonych rozkazów w czasie kompilacji. Na przykład rozkaz Move Characters (MVC) w S/390 powoduje przemieszczenie łańcucha znaków z jednej lokacji do drugiej. Za każdym razem, gdy jest on wykonywany, przesunięcie zależy od długości łańcucha, od tego, czy i w którym kierunku lokacje nakładają się, oraz jakie jest wyrównanie (*alignment*). W większości przypadków odpowiedzi na te pytania są znane w czasie kompilacji. Wobec tego kompilator mógłby tworzyć zoptymalizowaną sekwencję prymitywnych rozkazów dla tej funkcji.

Drugim, już odnotowanym punktem, jest to, że w każdym przypadku większość rozkazów generowanych przez kompilator stanowią rozkazy stosunkowo proste. Wydaje się rozsądne, że jednostka sterująca zbudowana specjalnie dla tych rozkazów i wykorzystująca niewiele (lub wcale) mikrorozkazów mogłaby wykonywać je szybciej niż porównywalna jednostka CISC.

Punkt trzeci odnosi się do potokowego przetwarzania rozkazów. Badacze architektury RISC przeczuwają, że metoda potokowego przetwarzania rozkazów może być stosowana bardziej efektywnie ze zredukowaną listą rozkazów. Przeanalizujemy teraz ten punkt bardziej szczegółowo.

Ostatnim i nieco mniej znaczącym punktem jest to, że programy RISC powinny lepiej reagować na przerwania, ponieważ przerwania są dokonywane najczęściej między operacjami elementarnymi. Architektury o złożonych rozkazach albo ograniczają przerwania do granic rozkazów, albo muszą określić specjalne punkty dopuszczalnych przerwań i wdrożyć mechanizmy powtórnego uruchomienia rozkazu.

Poprawa wydajności wynikająca z architektury o zredukowanej liście rozkazów jest daleka od udowodnienia. Przeprowadzono wiele badań, jednak nie na maszynach o porównywalnej technologii i mocy. Ponadto, w większości badań nie dążono do oddzielenia efektów zredukowanej listy rozkazów i efektów dużego zestawu rejestrów. „Poszlaki" są jednak sugestywne.

Drugi obszar potencjalnych korzyści, który jest wyraźniej zarysowany, wiąże się z wdrażaniem układów VLSI. Gdy stosuje się układy VLSI, projektowanie i imple-

mentacja procesora ulega podstawowym zmianom. Tradycyjny procesor, taki jak IBM S/390 bądź VAX, składa się z jednego lub wielu modułów drukowanych, zawierających znormalizowane układy SSI i MSI. Wraz z wynalezieniem układów LSI i VLSI stało się możliwe zmieszczenie całego procesora w jednym mikroukładzie. W przypadku jednoukładowego procesora występują dwa argumenty przemawiające za stosowaniem strategii RISC. Po pierwsze – zagadnienie wydajności: opóźnienia wewnątrz mikroukładu są o wiele mniejsze niż między mikroukładami. Racjonalne jest więc poświęcenie cennej powierzchni mikroukładu na implementacje tych operacji, które występują często. Widzieliśmy, że w rzeczywistości proste rozkazy i odniesienia do lokalnych skalarów są najczęstszymi działaniami. Mikroukłady RISC z Berkeley zostały zaprojektowane właśnie przy uwzględnieniu tych rozważań. Podczas gdy w typowym mikroprocesorze jednoukładowym około połowy powierzchni jest przeznaczone na pamięć sterującą zawierającą mikrorozkazy, w mikroukładzie RISC I jednostka sterująca zajmuje zaledwie około 6% powierzchni [SHER84].

Drugim zagadnieniem związanym z układami VLSI jest czas projektowania i wdrażania. Procesor VLSI jest trudny do opracowania. Zamiast polegać na dostępnych podzespołach SSI/MSI, projektant musi zaprojektować cały układ i jego strukturę już na poziomie podzespołu. W przypadku architektury o zredukowanej liście rozkazów proces ten jest daleko łatwiejszy, co widać w tabeli 13.7 [FITZ81]. Jeśli ponadto wydajność mikroukładu RISC będzie równoważna porównywalnym mikroprocesorom CISC, to zalety rozwiązania RISC stają się ewidentne.

Tabela 13.7. Pracochłonność projektowania układu i topologii wybranych mikroprocesorów

Procesor	Tranzystory (w tysiącach)	Projektowanie układu (w osobomiesiącach)	Projektowanie topologii (w osobomiesiącach)
RISC I	44	15	12
RISC II	41	18	12
M68000	68	100	70
Z8000	18	60	70
Intel iAPx-432	110	170	90

Własności CISC a własności RISC

Po początkowym entuzjazmie w stosunku do maszyn RISC zaczęło narastać przekonanie, że (1) projekty RISC mogą zyskać na włączeniu pewnych elementów CISC oraz że (2) projekty CISC mogą zyskać na włączeniu pewnych elementów RISC. W rezultacie nowsze projekty RISC, w tym PowerPC, nie są już „czystymi" RISC, a nowsze projekty CISC, w tym Pentium II i późniejsze modele Pentium, zawierają pewne własności RISC.

Interesujące porównanie zawarte w pracy [MASH95] daje pewien wgląd w to zagadnienie (tabela 13.8). W tabeli wymieniono pewną liczbę procesorów i porównano ich własności. Dla celów tego porównania przyjęto, że typowe dla RISC są następujące własności:

1. Rozkazy o jednakowej długości.
2. Długość ta wynosi zwykle 4 bajty.
3. Mała liczba trybów adresowania danych, zwykle mniejsza niż pięć. Parametr ten jest trudny do sprecyzowania. Tryby rejestrowy i znakowy nie są w tej tablicy liczone, natomiast różne formaty o różnych rozmiarach wyrównania są liczone oddzielnie.
4. Brak adresowania pośredniego, które wymaga jednego dostępu do pamięci w celu otrzymania adresu argumentu z pamięci.
5. Brak operacji łączących ładowanie/zapis z arytmetyką (np. dodaj z pamięci, dodaj do pamięci).
6. Co najwyżej jeden argument w rozkazie jest adresowany w pamięci.
7. Nie jest wykonywane dowolne wyrównywanie danych dla operacji ładowania/zapisu.
8. Maksymalna liczba zastosowań jednostki zarządzania pamięcią (MMU) w odniesieniu do adresu danych w rozkazie.
9. Liczba bitów specyfikatora rejestru liczb całkowitych jest równa 5 lub większa. Oznacza to, że przynajmniej 32 rejestry liczb całkowitych mogą być jednocześnie adresowane w sposób jawny.
10. Liczba bitów specyfikatora rejestru liczb zmiennopozycyjnych jest równa 4 lub większa. Oznacza to, że przynajmniej 16 rejestrów zmiennopozycyjnych może być jednocześnie adresowanych w sposób jawny.

Tabela 13.8. Własności wybranych procesorów

Procesor	Liczba rozmiarów rozkazów	Maksymalny rozmiar rozkazu (bajtów)	Liczba trybów adresowania	Adresowanie pośrednie	Ładuj/zapisz przy arytmetyce mieszanej	Maksymalna liczba argumentów pamięci	Dozwolone adresowanie nie wyrównane	Maksymalna liczba zastosowań mmu	Liczba bitów specyfikatora rejestru całkowitoliczbowego	Liczba bitów specyfikatora rejestru zmiennopozycyjnego
AMD29000	1	4	1	nie	nie	1	nie	1	8	3[a]
MIPS R2000	1	4	1	nie	nie	1	nie	1	5	4
SPARC	1	4	2	nie	nie	1	nie	1	5	4
MC88000	1	4	3	nie	nie	1	nie	1	5	4
HP PA	1	4	10[a]	nie	nie	1	nie	1	5	4
IBM RT/PC	2[a]	4	1	nie	nie	1	nie	1	4[a]	3[a]
IBM RS/6000	1	4	4	nie	nie	1	tak	1	5	5
Intel i860	1	4	4	nie	nie	1	nie	1	5	4
IBM 3090	4	8	2[b]	nie[b]	tak	2	tak	4	4	2
Intel 80486	12	12	15	nie[b]	tak	2	tak	4	3	3
NSC 32016	21	21	23	tak	tak	2	tak	4	3	3
MC68040	11	22	44	tak	tak	2	tak	8	4	3
VAX	56	56	22	tak	tak	6	tak	24	4	0
Clipper	4[a]	8[a]	9[a]	nie	nie	1	0	2	4[a]	3[a]
Intel 80960	2[a]	8[a]	9[a]	nie	nie	1	tak[a]	–	5	3[a]

[a] Procesor RISC, który nie ma tej własności.
[b] Procesor CISC, który nie ma tej własności.

Cechy od 1 do 3 są wskaźnikami złożoności dekodowania rozkazu. Cechy od 4 do 8 sugerują łatwość lub trudność przetwarzania potokowego, zwłaszcza wobec wymagań pamięci wirtualnej. Cechy 9 i 10 wiążą się ze zdolnością do wykorzystywania zalet kompilatorów.

Pierwszych 8 procesorów w tabeli ma w sposób oczywisty architekturę RISC, następnych 5 – oczywiście CISC, a ostatnie 2 procesory w rzeczywistości mają wiele cech CISC, chociaż są często uważane za RISC.

13.5. Przetwarzanie potokowe w architekturze RISC

Przetwarzanie potokowe rozkazów regularnych

Jak stwierdziliśmy w podrozdz. 12.4, w celu zwiększenia wydajności stosuje się często potokowe przetwarzanie rozkazów. Rozważmy to zagadnienie w kontekście architektury RISC. Większość rozkazów jest typu rejestr-rejestr, a cykl rozkazu ma następujące dwie fazy:

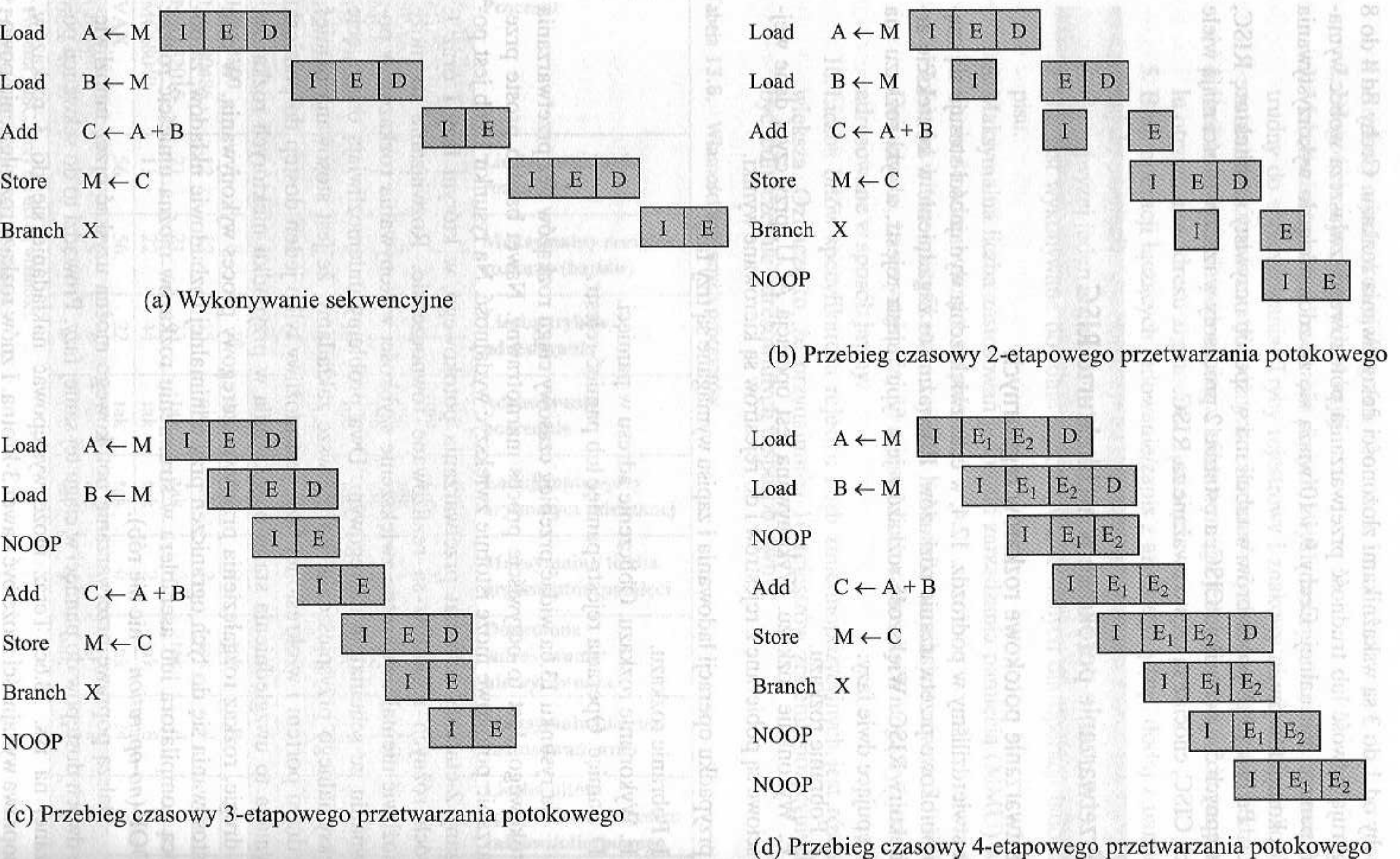
- I. Pobranie rozkazu.
- E. Wykonanie rozkazu. Wykonywana jest operacja ALU, przy czym dane wejściowe są pobierane z rejestrów i do rejestrów są kierowane wyniki.

W przypadku operacji ładowania i zapisu wymagane są trzy fazy:

- I. Pobranie rozkazu.
- E. Wykonanie rozkazu. Obliczenie adresu w pamięci.
- D. Pamięć. Operacja rejestr-pamięć lub pamięć-rejestr.

Na rysunku 13.6a widać przebieg czasowy ciągu rozkazów bez przetwarzania potokowego. Jest to oczywiście proces marnotrawny. Nawet bardzo proste przetwarzanie potokowe może istotnie zwiększyć wydajność. Na rysunku 13.6b jest pokazany 2-etapowy schemat przetwarzania potokowego, w którym fazy I oraz E dwóch różnych rozkazów są realizowane równocześnie. Rozwiązanie to może umożliwić niemal dwukrotne zwiększenie szybkości wykonywania rozkazów w porównaniu ze schematem szeregowym. Dwa problemy uniemożliwiają osiągnięcie maksymalnego przyspieszenia. Po pierwsze zakładamy, że jest stosowana pamięć z jednym portem i w określonej fazie jest możliwy tylko jeden dostęp do pamięci. Wymaga to uwzględnienia stanu oczekiwania w przypadku niektórych rozkazów. Po drugie, rozkaz rozgałęzienia przerywa szeregowy proces wykonywania. W celu dostosowania się do tych ograniczeń przy minimalnej rozbudowie układów, za pomocą kompilatora lub asemblera w strumieniu rozkazów można umieścić rozkaz NOOP (*no-operation* – nic nie rób).

Dalszą poprawę przetwarzania potokowego można uzyskać przez umożliwienie dwóch dostępów do pamięci w ciągu tej samej fazy. Prowadzi to do sekwencji pokazanej na rys. 13.6c. Teraz może występować nakładanie się do 3 rozkazów, a poprawa wydajności może być nawet 3-krotna. I znów rozkazy rozgałęzienia powo-



Rysunek 13.6. Efekty przetwarzania potokowego

dują niepełne wykorzystanie możliwości przyspieszenia. Zauważmy, że zależności między danymi również wywierają pewien wpływ. Jeśli rozkaz wymaga argumentu, który został zmieniony przez poprzedni rozkaz, jest potrzebne opóźnienie. Podobnie jak poprzednio, można się do tego dostosować za pomocą rozkazu NOOP.

Przetwarzanie potokowe przedyskutowane dotychczas funkcjonuje najlepiej, jeśli trzy fazy trwają w przybliżeniu tyle samo. Ponieważ faza E obejmuje zwykle operację ALU, może ona być dłuższa. W tym przypadku możemy ją podzielić na dwie podfazy:

- E_1. Odczyt pliku z rejestru.
- E_2. Operacja ALU i zapis w rejestrze.

Ze względu na prostotę i regularność listy rozkazów, zaprojektowanie schematu 3- lub 4-fazowego jest łatwe. Na rysunku 13.6d widać potok 4-etapowy. W określonej chwili w trakcie wykonywania może się znajdować do 4 rozkazów, co daje maksymalny potencjalny czynnik przyspieszenia równy 4. Zauważmy, że w celu dostosowania się do opóźnień spowodowanych przez rozgałęzianie i zależność danych, znów są potrzebne rozkazy NOOP.

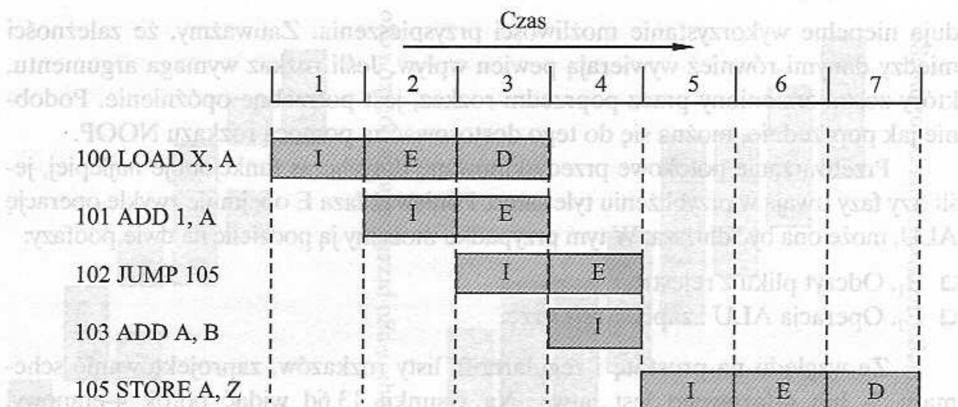
Optymalizacja przetwarzania potokowego

Dzięki prostocie i regularności rozkazów RISC schematy przetwarzania potokowego mogą być efektywnie stosowane. Istnieje pewne zróżnicowanie czasu wykonywania rozkazów i jest możliwe przystosowanie potoku do tej sytuacji. Jak jednak widzieliśmy, zależność danych i rozgałęzianie redukują ogólną szybkość wykonywania.

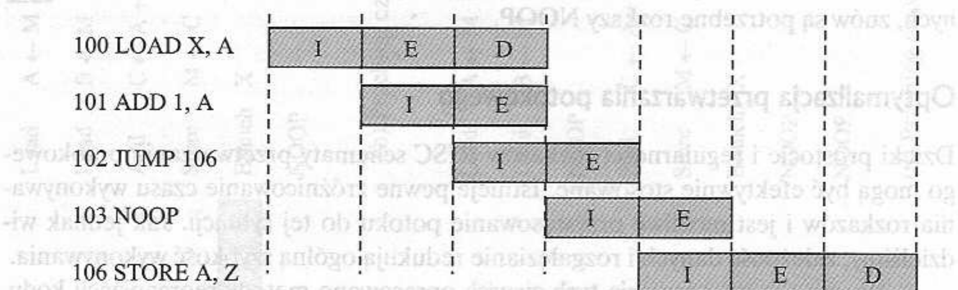
W celu skompensowania tych zjawisk opracowano metody reorganizacji kodu. Rozważmy najpierw rozkazy rozgałęziania. Przy *rozgałęzianiu opóźnionym* jako metodzie zwiększania efektywności potoku używa się rozgałęzienia, które nie daje wyniku aż do zakończenia następnego rozkazu (Stąd określenie „opóźnione"). Lokacja rozkazu następującego bezpośrednio po rozgałęzieniu jest określona jako *szczelina opóźnienia*. Ta dziwna procedura jest zilustrowana w tabeli 13.9. W pierwszej kolumnie widzimy zwyczajny, symboliczny rozkaz w języku maszynowym. Po wykonaniu rozkazu 102 następnym rozkazem przewidzianym do wykonania jest 105. W celu zapewnienia płynności potoku po tym rozkazie jest wstawiany rozkaz NOOP. Można jednak zwiększyć wydajność, zamieniając miejscami rozkazy 101 i 102.

Tabela 13.9. Rozgałęzienie normalne i opóźnione

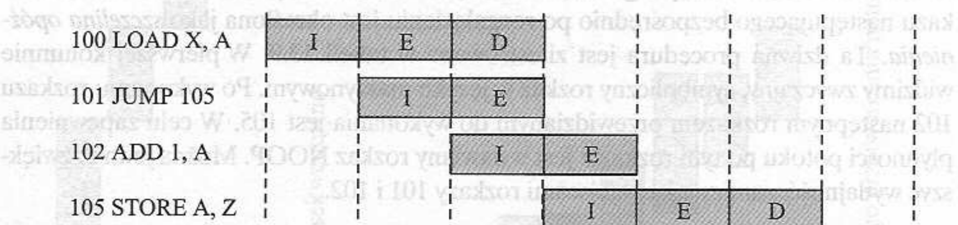
Adres	Rozgałęzienie normalne		Rozgałęzienie opóźnione		Zoptymalizowane rozgałęzienie opóźnione	
100	LOAD	X,A	LOAD	X,A	LOAD	X,A
101	ADD	1,A	ADD	1,A	JUMP	105
102	JUMP	105	JUMP	106	ADD	1,A
103	ADD	A,B	NOOP		ADD	A,B
104	SUB	C,B	ADD	A,B	SUB	C,B
105	STORE	A,Z	SUB	C,B	STORE	A,Z
106			STORE	A,Z		



(a) Potok tradycyjny



(b) Wstawiony rozkaz pusty (NOOP)



(c) Rozkazy odwrócone

Rysunek 13.7. Zastosowanie opóźnionego rozgałęziania

Wynik jest pokazany na rys. 13.7. Na rysunku 13.7a pokazano tradycyjne podejście do przetwarzania potokowego, które zostało przeanalizowane w rozdz. 12. (zobacz np. rys. 12.11 i 12.12). W chwili 3 jest pobierany rozkaz JUMP (skok). W chwili 4 rozkaz JUMP jest wykonywany, a jednocześnie jest pobierany rozkaz 103 (ADD, dodaj). Ponieważ następuje skok i stan licznika rozkazów jest aktualizowany, rozkaz 103 musi być usunięty z potoku; w chwili 5 jest ładowany rozkaz 105, bę-

dący celem skoku. Na rysunku 13.7b pokazano ten sam potok przetwarzany w typowej organizacji RISC. Przebieg czasowy jest taki sam, jednak ze względu na wstawienie rozkazu NOOP nie potrzebujemy specjalnych układów do oczyszczania potoku; jest po prostu wykonywany rozkaz NOOP bez jakiegokolwiek efektu. Na rysunku 13.7c pokazano zastosowanie rozgałęzienia opóźnionego. Rozkaz JUMP jest pobierany w chwili 2, przed rozkazem ADD, który jest pobierany w chwili 3. Zauważmy jednak, że rozkaz ADD jest pobierany, zanim wykonywanie rozkazu JUMP ma szansę zmienić stan licznika programu. Wobec tego w chwili 4 następuje jednocześnie wykonywanie rozkazu ADD i pobieranie rozkazu 105. Oryginalna semantyka programu jest więc zachowana, jednak wykonywanie rozkazów wymaga czasu krótszego o jeden cykl.

Taka zamiana rozkazów jest skuteczna w przypadku rozgałęzień bezwarunkowych, wywołań i powrotów. W odniesieniu do rozgałęzień warunkowych procedura ta nie może być stosowana na ślepo. Jeśli warunek testowany w ramach rozgałęzienia może być zmieniony przez rozkaz bezpośrednio poprzedzający, to kompilator musi powstrzymać się od zamiany rozkazów, a zamiast tego wstawić rozkaz NOOP. Doświadczenie z systemami Berkeley RISC oraz IBM 801 wskazuje, że większość rozkazów rozgałęzienia warunkowego może być w ten sposób zoptymalizowana [PATT82a, RADI83].

Podobna taktyka, znana jako *opóźnione ładowanie*, może być zastosowana w odniesieniu do rozkazów LOAD (ładowania). W przypadku rozkazów LOAD rejestr docelowy tej operacji jest blokowany przez procesor. Następnie procesor kontynuuje wykonywanie strumienia rozkazów aż do osiągnięcia rozkazu wymagającego tego rejestru, po czym pozostaje bezczynny do końca ładowania. Jeśli kompilator może tak zmienić porządek rozkazów, żeby podczas przebywania ładowania w potoku mogła być wykonywana użyteczna praca, to efektywność wzrasta.

Na zakończenie zauważmy, że projektowanie potoku rozkazów nie może być prowadzone w oderwaniu od innych zabiegów optymalizacyjnych stosowanych w systemie. W pracy [BRAD91b] wskazano na przykład, że szeregowanie rozkazów w potoku oraz dynamiczne przydzielanie rejestrów muszą być rozpatrywane łącznie, co umożliwia osiągnięcie największej efektywności.

13.6. MIPS R4000

Jeden z pierwszych dostępnych na rynku zestawów mikroukładów RISC został opracowany w MIPS Technology Inc. Inspiracją dla niego był system eksperymentalny również noszący nazwę MIPS, opracowany w Stanfordzie [HENN84]. Najnowszym członkiem rodziny MIPS jest procesor R4000. Ma on zasadniczo tę samą architekturę i listę rozkazów, co wcześniejsze modele MIPS: R2000, R3000. Najbardziej znaczącą różnicą jest to, że w R4000 wszystkie wewnętrzne i zewnętrzne ścieżki danych, a także adresy, rejestry i ALU są 64-, a nie 32-bitowe.

Zastosowanie architektury 64-bitowej zamiast 32-bitowej ma wiele zalet. Większa jest przestrzeń adresowa – wystarczająco, aby system operacyjny odwzorował

ponad terabajt plików bezpośrednio w pamięci wirtualnej w celu ułatwienia dostępu. Ponieważ obecnie powszechne są napędy dyskowe o pojemności 1 GB i większej, 4-gigabajtowa przestrzeń adresowa maszyny 32-bitowej staje się ograniczeniem. Dzięki architekturze 64-bitowej procesor R4000 może przetwarzać za jednym razem takie dane, jak liczby zmiennopozycyjne IEEE o podwójnej dokładności lub łańcuchy znaków obejmujące do 8 znaków.

Mikroukład procesora R4000 składa się z dwóch sekcji, z których jedna zawiera procesor, a druga koprocesor do zarządzania pamięcią. Procesor ma bardzo prostą architekturę. Intencją było zaprojektowanie systemu, w którym układy logiczne wykonywania rozkazów byłyby maksymalnie proste, co pozostawiałoby miejsce na układy logiczne zwiększające wydajność (np. na całą jednostkę zarządzania pamięcią).

Procesor obsługuje 32 rejestry 64-bitowe. Zawiera również do 128 KB szybkiej pamięci podręcznej, w połowie przeznaczonej na rozkazy, a w połowie na dane. Stosunkowo duża pamięć podręczna (IBM 3090 obsługuje 128÷256 KB pamięci podręcznej) pozwala systemowi na trzymanie dużych zbiorów kodów programu i danych lokalnych wobec procesora, na rozładowywanie magistrali pamięci głównej oraz na uniknięcie dużej tablicy rejestrów i towarzyszących jej układów logicznych okienkowania.

Lista rozkazów

W tabeli 13.10 są wymienione podstawowe rozkazy wszystkich procesorów MIPS serii R, a w tabeli 13.11 są podane dodatkowe rozkazy zaimplementowane w R4000. Wszystkie rozkazy procesora są zakodowane w jednym formacie 32-bitowym. Wszystkie operacje na danych są typu rejestr-rejestr; jedynie odniesienia do pamięci są czystymi operacjami ładowania/zapisu.

Procesor R4000 nie używa kodów warunkowych. Jeśli rozkaz generuje warunek, to odpowiednie znaczniki stanu są przechowywane w rejestrze ogólnego przeznaczenia. Dzięki temu uniknięto specjalnych układów logicznych zajmujących się kodami warunkowymi, które oddziałują na mechanizm przetwarzania potokowego oraz na zmienianie kolejności rozkazów przez kompilator. Zamiast tego wykorzystano już istniejący mechanizm wdrożony w celu zajmowania się zależnościami danych rejestrowych. Ponadto, warunki odwzorowane w tablicach rejestrów podlegają takiej samej optymalizacji w czasie kompilowania obejmującej rozmieszczanie i powtórne użycie, jak inne wartości przechowywane w rejestrach.

Podobnie jak większość maszyn RISC, procesor MIPS używa rozkazu o pojedynczej, 32-bitowej długości. Upraszcza to pobieranie i dekodowanie rozkazów, a także upraszcza współpracę z jednostką zarządzania pamięcią wirtualną podczas pobierania rozkazów (rozkazy nie przekraczają granic słów lub stron). Zastosowano trzy formaty rozkazów (rys. 13.8) ze wspólnym formatowaniem kodów operacji i odniesień do rejestrów, co upraszcza dekodowanie rozkazów. Wyniki bardziej złożonych rozkazów mogą być syntetyzowane w czasie kompilacji.

Tabela 13.10. Lista rozkazów procesorów MIPS serii R

Operacja	Opis	Operacja	Opis
Rozkazy ładowania/zapisu		**Rozkazy mnożenia/dzielenia**	
LB	Ładuj bajt	MULT	Pomnóż
LBU	Ładuj bajt bez znaku	MULTU	Pomnóż bez znaku
LH	Ładuj półsłowo	DIV	Podziel
LHU	Ładuj półsłowo bez znaku	DIVU	Podziel bez znaku
LW	Ładuj słowo	MFHI	Przenieś z HI
LWL	Ładuj słowo w lewo	MTHI	Przenieś do HI
LWR	Ładuj słowo w prawo	MFLO	Przenieś z LO
SB	Zapisz bajt	MTLO	Przenieś do LO
SH	Zapisz półsłowo	**Rozkazy skoku i rozgałęzienia**	
SW	Zapisz słowo	J	Skocz
SWL	Zapisz słowo w lewo	JAL	Skocz i połącz
SWR	Zapisz słowo w prawo	JR	Skocz do rejestru
Rozkazy arytmetyczne (argument natychmiastowy w ALU)		JALR	Skocz i połącz rejestr
ADDI	Dodaj natychmiastowy	BEQ	Rozgałęziaj, gdy równe
ADDIU	Dodaj natychmiastowy bez znaku	BNE	Rozgałęziaj, gdy nierówne
SLTI	Ustaw na mniej niż natychmiastowy	BLEZ	Rozgałęziaj, gdy mniejsza lub równa zeru
SLTIU	Ustaw na mniej niż natychmiastowy bez znaku	BGTZ	Rozgałęziaj, gdy większa od zera
ANDI	AND z argumentem natychmiastowym	BLTZ	Rozgałęziaj, gdy mniejsza od zera
ORI	OR z argumentem natychmiastowym	BGEZ	Rozgałęziaj, gdy większa lub równa zeru
XORI	Exclusive-OR z argumentem natychmiastowym	BLTZAL	Rozgałęziaj, gdy mniejsza od zera i łącznika
LUI	Ładuj argument natychmiastowy do górnej połowy rejestru	BGEZAL	Rozgałęziaj, gdy większa lub równa zeru i łącznikowi
Rozkazy arytmetyczne (3-argumentowe, typu R)		**Rozkazy dotyczące koprocesora**	
ADD	Dodaj	LWCz	Ładuj słowo do koprocesora
ADDU	Dodaj bez znaku	SWCz	Zapisz słowo w koprocesorze
SUB	Odejmij	MTCz	Przenieś do koprocesora
SUBU	Odejmij bez znaku	MFCz	Przenieś z koprocesora
SLT	Ustaw na mniej niż	CTCz	Przenieś sterowanie do koprocesora
SLTU	Ustaw na mniej niż bez znaku	CFCz	Przenieś sterowanie z koprocesora
AND	Wykonaj AND	COPz	Operacja koprocesora
OR	Wykonaj OR	BCzT	Rozgałęziaj, gdy „z” koprocesora prawdziwe
XOR	Wykonaj XOR	BCzF	Rozgałęziaj, gdy „z” koprocesora fałszywe
NOR	Wykonaj NOR	**Rozkazy specjalne**	
Rozkazy przesunięcia		SYSCALL	Wywołanie systemowe
SLL	Przesuń logicznie w lewo	BREAK	Przerwanie
SRL	Przesuń logicznie w prawo		
SRA	Przesuń arytmetycznie w prawo		
SLLV	Przesuń zmienną logicznie w lewo		
SRLV	Przesuń zmienną logicznie w prawo		
SRAV	Przesuń zmienną arytmetycznie w prawo		

Tabela 13.11. Dodatkowe rozkazy R4000

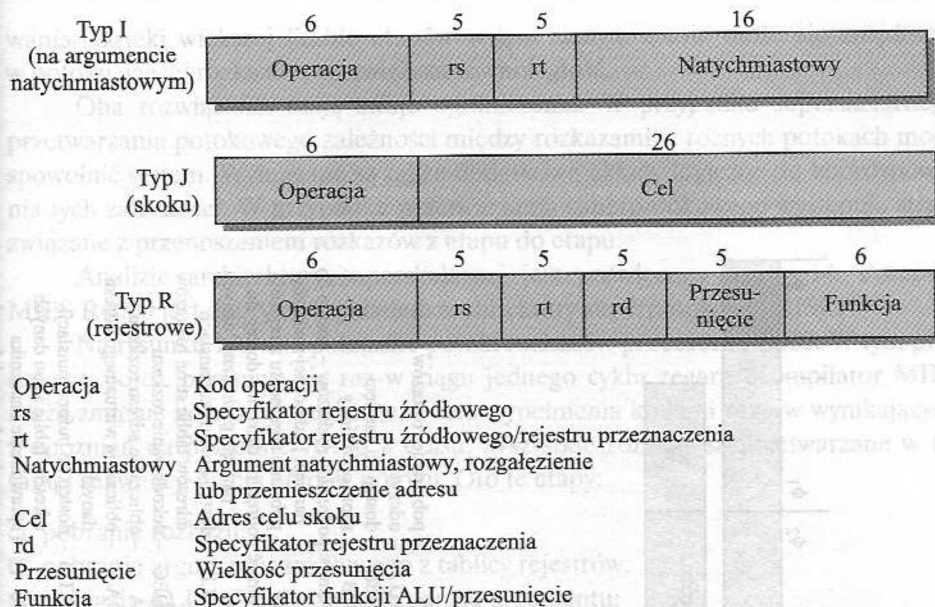
Operacja	Opis	Operacja	Opis
Rozkazy ładowania/zapisu		**Rozkazy wyjątku**	
LL	Ładuj połączony	TGE	Przerwij (*trap*), jeśli większa niż lub równa
S.C.	Zapisz warunkowy	TGEU	Przerwij, jeśli większa niż lub równa bez znaku
SYNC	Synchronizacja	TLT	Przerwij, jeśli mniejsza niż
Rozkazy skoku i rozgałęzienia		TLTU	Przerwij, jeśli mniejsza niż bez znaku
BEQL	Rozgałęziaj, gdy prawdopodobnie równa	TEQ	Przerwij, jeśli równa
BNEL	Rozgałęziaj, gdy prawdopodobnie nierówna	TNE	Przerwij, jeśli nie równa
BLEZL	Rozgałęziaj, gdy prawdopodobnie mniejsza lub równa zeru	TGEI	Przerwij, jeśli większa lub równa natychmiastowemu
BGTZL	Rozgałęziaj, gdy prawdopodobnie większa niż zero	TGEIU	Przerwij, jeśli większa lub równa natychmiastowemu bez znaku
BLTZL	Rozgałęziaj, gdy prawdopodobnie mniejsza od lub równa zeru	TLTI	Przerwij, jeśli mniejsza niż natychmiastowy
BGEZL	Rozgałęziaj, gdy prawdopodobnie większa od lub równa zeru	TLTIU	Przerwij, jeśli mniejsza niż natychmiastowy bez znaku
BLTZAL	Rozgałęziaj, gdy prawdopodobnie mniejsza od zera i łącznika	TEQI	Przerwij, jeśli równa natychmiastowemu
BGEZAL	Rozgałęziaj, gdy prawdopodobnie większa lub równa od zera i łącznika	TNEI	Przerwij, jeśli nie równa natychmiastowemu
BCzTL	Rozgałęziaj, gdy prawdopodobnie „z" koprocesora prawdziwe	**Rozkazy dotyczące koprocesora**	
CDzFL	Rozgałęziaj, gdy prawdopodobnie „z" koprocesora fałszywe	LDCz	Ładuj podwójne do koprocesora
		SDCz	Zapisz podwójne w koprocesorze

Sprzętowo zaimplementowano jedynie najprostszy i najczęściej używany tryb adresowania. Wszystkie odniesienia do pamięci obejmują 16-bitowe wyrównanie w stosunku do rejestru 32-bitowego. Na przykład rozkaz „ładuj słowo" ma postać:

```
lw r2, 128(r3) ładuj słowo zawarte pod adresem 128 względem
rejestru 3 do rejestru 2
```

Każdy z 32 rejestrów ogólnego przeznaczenia może być użyty jako rejestr podstawowy. Jeden rejestr, a mianowicie r0, zawsze zawiera zera.

Kompilator używa wielu rozkazów maszynowych w celu syntetyzowania trybów adresowania typowych dla maszyn konwencjonalnych. Pewne przykłady zawiera tabela 13.12 [CHOW87]. Pokazano w niej użycie rozkazu lui (*load upper immediate*). Rozkaz ten powoduje załadowanie 16-bitowej wartości natychmiastowej do górnej połowy rejestru i wyzerowanie dolnej połowy.



Rysunek 13.8. Formaty rozkazów MIPS

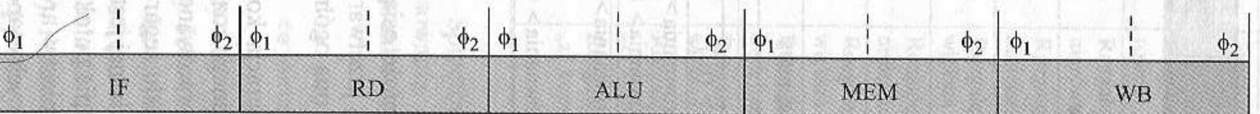
Tabela 13.12. Synteza pozostałych trybów adresowania za pomocą trybu adresowania MIPS

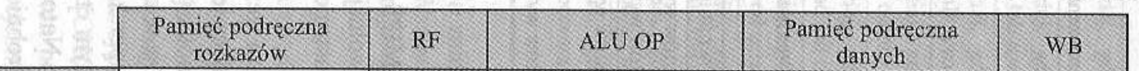
Rozkaz jawny	Rozkaz rzeczywisty
lw r2, <16-bitowe wyrównanie>	lw r2, <16-bitowe wyrównanie> (r0)
lw r2, <32-bitowe wyrównanie>	lui r1, < 16 wyższych bitów wyrównania> lw r2, < 16 niższych bitów wyrównania> (r1)
lw r2, <32-bitowe wyrównanie> (r4)	lui r1, < 16 wyższych bitów wyrównania> addu r1, r1, r4 lw r2, < 16 niższych bitów wyrównania> (r1)

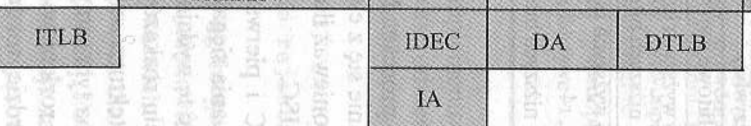
Potok rozkazów

Dzięki uproszczonej architekturze rozkazów procesor MIPS może osiągać bardzo efektywne przetwarzanie potokowe. Zapoznanie się z ewolucją przetwarzania potokowego procesorów MIPS jest pouczające, ponieważ ilustruje ona ogólną ewolucję przetwarzania potokowego w architekturze RISC.

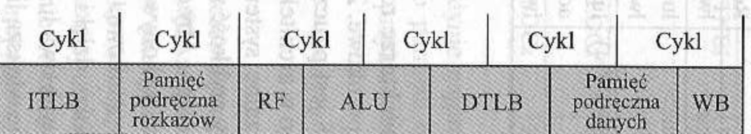
Pierwsze, doświadczalne systemy RISC i pierwsza generacja komercyjnych procesorów RISC osiągały szybkość wykonywania sięgającą jednego rozkazu w jednym cyklu zegara systemowego. Aby zwiększyć tę wydajność, opracowano dwie klasy procesorów, umożliwiające wykonywanie wielu rozkazów w cyklu zegara: architektury superskalarną i superpotokową. W architekturze superskalarnej jest w zasadzie powielany każdy etap potoku, dzięki czemu na tym samym etapie potoku może być jednocześnie przetwarzanych dwa lub więcej rozkazów. Natomiast w architekturze superpotokowej stosuje się większą liczbę bardziej rozdrobnionych etapów potoko-



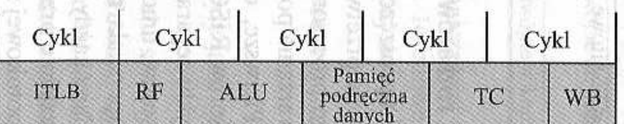




(a) Uszczegółowiony potok R3000



(b) Zmodyfikowany potok R3000 zredukowanych czasów oczekiwania



(c) Zoptymalizowany potok R3000 z równoległym dostępem do TLB (bufora translacji adresów tablic stron) oraz do pamięci podręcznej

IF – pobieranie rozkazów
RD – odczyt
MEM – dostęp do pamięci
WB – zapis opóźniony
Pamięć podręczna rozkazów – dostęp do pamięci podręcznej rozkazów
RF – pobranie argumentu z rejestru
Pamięć podręczna danych – dostęp do pamięci podręcznej danych
ITLB – dostęp do bufora translacji adresów tablic stron rozkazów
IDEC – dekodowanie rozkazu
IA – obliczanie adresu rozkazu
DA – obliczanie wirtualnego adresu danych
DTLB – dostęp do bufora translacji adresów tablic stron danych
TC – sprawdzenie znacznika pamięci podręcznej danych

Rysunek 13.9. Wzbogacenie potoku R3000

wania. Dzięki większej liczbie etapów w tym samym czasie może się znajdować w potoku więcej rozkazów, co zwiększa równoległość.

Oba rozwiązania mają swoje ograniczenia. W przypadku superskalarnego przetwarzania potokowego zależności między rozkazami w różnych potokach mogą spowolnić system. Wymagane są także dodatkowe układy logiczne do koordynowania tych zależności. W przypadku przetwarzania superpotokowego występują straty związane z przenoszeniem rozkazów z etapu do etapu.

Analizie architektury superskalarnej jest poświęcony rozdz. 14. Procesor MIPS R4000 jest dobrym przykładem architektury superpotokowej RISC.

Na rysunku 13.9a są pokazane potoki rozkazów procesora R3000. W tym procesorze potok przesuwa się raz w ciągu jednego cyklu zegara. Kompilator MIPS może zmienić kolejność rozkazów w celu wypełnienia kodami przerw wynikających z opóźnień na poziomie 70÷90% czasu. Wszystkie rozkazy są przetwarzane w tej samej sekwencji pięciu etapów potoku. Oto te etapy:

- pobranie rozkazu;
- pobranie argumentu źródłowego z tablicy rejestrów;
- operacja ALU lub generowanie adresu argumentu;
- odniesienie do danych w pamięci;
- zapis w tablicy rejestrów.

Jak widać na rys. 13.9a, występuje tutaj nie tylko równoległość wynikająca z przetwarzania potokowego, ale także równoległość w ramach wykonywania pojedynczego rozkazu. Cykl zegara o długości 60 ns jest podzielony na dwie fazy 30-nanosekundowe. Operacje dostępu do rozkazu zewnętrznego i do danych w pamięci podręcznej wymagają (każda z nich) 60 ns, podobnie jak główne operacje wewnętrzne (OP, DA, IA). Dekodowanie rozkazu jest operacją prostszą, wymagającą tylko jednej fazy 30 ns, nakładającej się z pobraniem zawartości rejestru w ramach tego samego rozkazu. Obliczanie adresu w ramach rozkazu rozgałęzienia również nakłada się z dekodowaniem rozkazu i pobraniem zawartości rejestru, tak więc rozgałęzienie przy rozkazie i może adresować dostęp ICACHE do rozkazu $i + 2$. Podobnie ładowanie rozkazu i powoduje pobranie danych, które są natychmiast używane przez kod operacji rozkazu $i + 1$, podczas gdy wynik operacji ALU lub przesunięcia jest bez opóźnienia kierowany do rozkazu $i + 1$. To ścisłe sprzężenie między rozkazami umożliwia wysoce wydajne przetwarzanie potokowe.

Każdy cykl zegara jest dzielony na dwa odrębne etapy, oznaczane jako $\phi 1$ i $\phi 2$. Funkcje realizowane w każdej fazie są zebrane w tabeli 13.13.

Procesor R4000 zawiera wiele udoskonaleń technicznych w porównaniu z R3000. Zastosowanie bardziej zaawansowanej technologii pozwala na skrócenie cyklu zegara do połowy, tzn. do 30 ns, oraz na podobne skrócenie czasu dostępu do tablicy rejestrów. Ponadto większa jest gęstość upakowania elementów w mikroukładzie, co pozwoliło na włączenie do tego samego mikroukładu pamięci podręcznych rozkazów i danych. Zanim przeanalizujemy przetwarzanie potokowe w procesorze R4000, rozważmy, jak można poprawić wydajność potoku procesora R3000 za pomocą techniki wykorzystywanej w R4000.

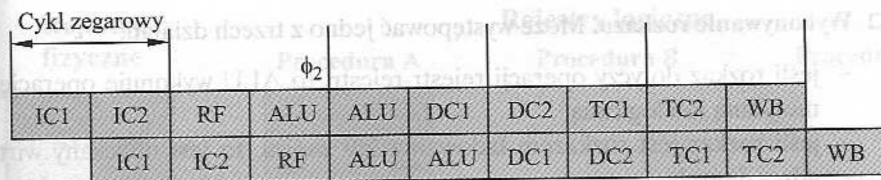
Tabela 13.13. Etapy potoku R3000

Etap potoku	Faza	Działanie
IF	$\phi 1$	Używając TLB, przekształć adres wirtualny rozkazu na adres fizyczny (po decyzji rozgałęzienia)
IF	$\phi 2$	Wyślij adres fizyczny do (rejestru) adresu rozkazu
RD	$\phi 1$	Spowoduj powrót rozkazu z pamięci podręcznej rozkazów Porównaj znaczniki i ważność pobranego rozkazu
RD	$\phi 2$	Dekoduj rozkaz Odczytaj tablicę rejestrów Jeśli następuje rozgałęzienie, oblicz adres docelowy rozgałęzienia
ALU	$\phi 1 + \phi 2$	Jeśli następuje operacja rejestr-rejestr, wykonywana jest operacja arytmetyczna lub logiczna
ALU	$\phi 1$	Jeśli następuje rozgałęzienie, decyduj, czy jest ono wykonywane Jeśli następuje odniesienie do pamięci (ładowanie lub zapis), oblicz wirtualny adres danych
ALU	$\phi 2$	Jeśli następuje odniesienie do pamięci, przekształć adres wirtualny danych na adres fizyczny, posługując się TLB
MEM	$\phi 1$	Jeśli następuje odniesienie do pamięci, wyślij adres fizyczny do pamięci podręcznej danych
MEM	$\phi 2$	Jeśli następuje odniesienie do pamięci, spowoduj powrót danych z pamięci podręcznej danych i sprawdź znaczniki
WB	$\phi 1$	Zapisz w tablicy rejestrów

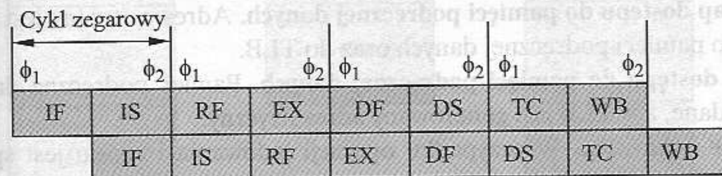
Pierwszy krok jest pokazany na rys. 13.9b. Pamiętajmy, że cykle na tym rysunku zajmują połowę czasu cykli z rys. 13.9a. Dzięki włączeniu do tego samego mikroukładu wszystkich modułów procesora, etapy pamięci podręcznych rozkazów i danych są krótsze o połowę; mogą więc one nadal zajmować tylko jeden cykl zegara. Podobnie, ponieważ przyspieszono dostęp do tablicy rejestrów, zapis i odczyt w rejestrze nadal zajmują połowę cyklu zegara.

Ponieważ pamięci podręczne procesora R4000 zostały scalone w tym samym mikroukładzie, proces tłumaczenia adresu wirtualnego na fizyczny może opóźniać dostęp do pamięci podręcznej. Opóźnienie to jest redukowane przez wdrożenie wirtualnie indeksowanych pamięci podręcznych oraz wprowadzenie równoległego dostępu do pamięci podręcznej i translacji adresu. Na rysunku 13.9c jest pokazany zoptymalizowany potok R3000 z wyżej opisanymi ulepszeniami. Ze względu na kompresję zdarzeń sprawdzanie wskaźników pamięci podręcznej danych jest prowadzone oddzielnie w cyklu następującym po dostępie do pamięci podręcznej.

W systemie superpotokowym te same układy elektroniczne są używane kilkakrotnie w czasie jednego cyklu dzięki wstawieniu rejestrów potoku rozszczepiających każdy etap. W zasadzie każdy etap superpotoku przebiega z wielokrotnością częstotliwości zegara, przy czym wielokrotność zależy od stopnia superpotokowania. Technologia R4000 dysponuje szybkością i gęstością, które umożliwiają superpotokowanie w stopniu 2. Na rysunku 13.10a jest pokazany zoptymalizowany potok R3000



(a) Superpotokowe wdrożenie zoptymalizowanego potoku R3000



IF – pobieranie rozkazu – pierwsza połowa
IS – pobieranie rozkazu – druga połowa
RF – pobranie argumentu z rejestru
EX – wykonywanie rozkazów
IC – pamięć podręczna rozkazów
DC – pamięć podręczna danych
DF – pamięć podręczna danych – pierwsza połowa
DS – pamięć podręczna danych – druga połowa
TC – sprawdzenie znacznika

(b) Potok R4000

Rysunek 13.10. Teoretyczny superpotok R3000 i rzeczywisty superpotok R4000

wykorzystujący to superpotokowanie. Zauważmy, że jest to w zasadzie taka sama struktura dynamiczna jak na rys. 13.9c.

Mogą być poczynione dalsze ulepszenia. Dla procesora R4000 zaprojektowano o wiele większy i wyspecjalizowany sumator. Umożliwia to prowadzenie operacji ALU z 2-krotnie większą szybkością. Inne ulepszenia pozwalają na 2-krotne przyspieszenie operacji ładowania i zapisu. Otrzymany w wyniku tych zabiegów potok jest pokazany na rys. 13.10b.

Potok procesora R4000 ma 8 etapów, co oznacza, że w potoku może się jednocześnie znajdować do 8 rozkazów. Potok przesuwa się z szybkością 2 etapów w jednym cyklu zegara. Etapy potoku są następujące:

- **Pobieranie rozkazu – pierwsza połowa.** Adres wirtualny jest przekazywany do pamięci podręcznej rozkazów oraz do bufora translacji adresów tablic stron TLB (*translation lookaside buffer*).
- **Pobieranie rozkazu – druga połowa.** Pamięć podręczna rozkazu przekazuje rozkaz, a w TLB jest generowany adres fizyczny.
- **Tablica rejestrów.** Równolegle następują trzy działania:
  - dekodowany jest rozkaz i dokonywane jest sprawdzenie warunków współzależności (tzn. sprawdzenie, czy rozkaz ten zależy od wyniku poprzedniego rozkazu);
  - sprawdzany jest wskaźnik pamięci podręcznej rozkazów;
  - z tablicy rejestrów są pobierane argumenty.

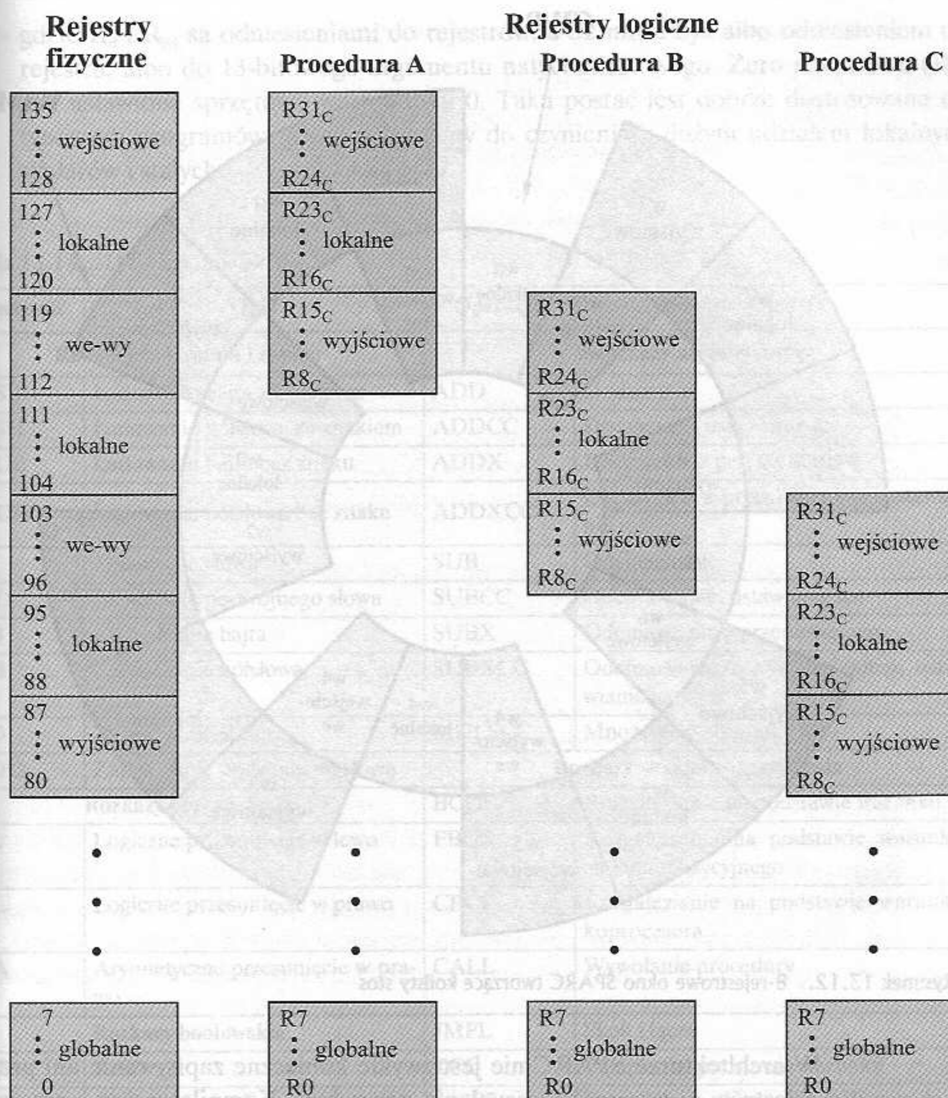
- **Wykonywanie rozkazu.** Może występować jedno z trzech działań:
  - jeśli rozkaz dotyczy operacji rejestr-rejestr, to ALU wykonuje operację arytmetyczną lub logiczną;
  - jeśli rozkaz jest rozkazem ładowania lub zapisu, to jest obliczany wirtualny adres danych;
  - jeśli rozkaz jest rozgałęzieniem, to jest obliczany wirtualny adres docelowy i są sprawdzane warunki rozgałęzienia.
- **Pierwszy etap dostępu do pamięci podręcznej danych.** Adres wirtualny jest przekazywany do pamięci podręcznej danych oraz do TLB.
- **Drugi etap dostępu do pamięci podręcznej danych.** Pamięć podręczna danych przekazuje dane, a w TLB jest generowany adres fizyczny.
- **Sprawdzenie wskaźnika.** W przypadku operacji ładowania i zapisu jest sprawdzany wskaźnik pamięci podręcznej.
- **Zapis.** Wynik jest zapisywany w tablicy rejestrów.

13.7. SPARC

SPARC (*Scalable Processor Architecture* – skalowalna architektura procesora) odnosi się do architektury stworzonej w Sun Microsystems. W firmie Sun powstały własne implementacje SPARC; niezależnie od tego firma ta udziela licencji innym producentom, zainteresowanym wytwarzaniem maszyn kompatybilnych z mikroprocesorem SPARC. Architektura ta została zainspirowana przez komputer RISC I powstały w Berkeley, zaś lista rozkazów i organizacja rejestrów są oparte ściśle na tym właśnie modelu.

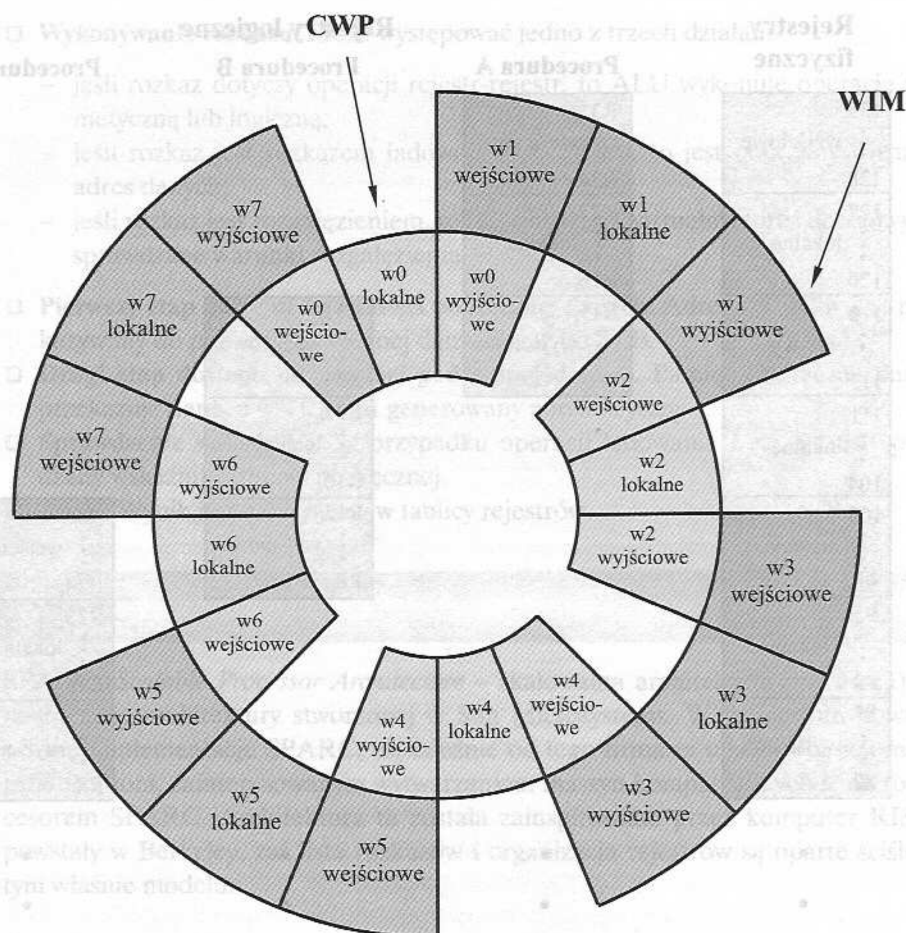
Zbiór rejestrów SPARC

Podobnie jak w komputerze RISC z Berkeley, w SPARC zostały zastosowane okna rejestrów. Każde okno składa się z 24 rejestrów, a całkowita liczba okien – zależna od implementacji – sięga od 2 do 32. Na rysunku 13.11 została pokazana implementacja obsługująca osiem okien, ze 136 rejestrami fizycznymi; jak wykazała analiza przeprowadzona w podrozdz. 13.2, wydaje się to być rozsądną liczbą okien. Rejestry fizyczne od 0 do 7 są rejestrami globalnymi, użytkowanymi wspólnie przez wszystkie procedury. Każdy proces widzi rejestry logiczne od 0 do 31. Rejestry logiczne od 24 do 31 – określane jako *wejściowe* – są używane wspólnie z procedurą wywołującą (macierzystą), natomiast rejestry logiczne od 8 do 15 – określane jako *wyjściowe* – są używane wspólnie z dowolną procedurą wywoływaną. Obie te części nakładają się z innymi oknami. Rejestry logiczne od 16 do 23 – określane jako *lokalne* – nie są współużytkowane ani nie nakładają się z innymi oknami. Jak wynika z dyskusji przeprowadzonej w podrozdz. 12.1, dostępność 8 rejestrów służących przekazywaniu parametrów powinna być w większości przypadków wystarczająca (patrz np. tabela 13.4).



Rysunek 13.11. Układ okien rejestrów SPARC o trzech procedurach

Na rysunku 13.12 zostało pokazane inne ujęcie nakładania się rejestrów. Procedura wywołująca umieszcza dowolne parametry, które mają być przekazane, w rejestrach wyjściowych; natomiast procedura wywoływana traktuje te same rejestry fizyczne jako wejściowe. Procesor prowadzi *bieżący wskaźnik okna* (*current window pointer* – CWP) znajdujący się w *rejestrze stanu procesora* (*processor status register* – PSR) i wskazujący okno aktualnie realizowanej procedury. *Maska nieważności okna* (*window invalid mask* – WIM), również znajdująca się w PSR, wskazuje, które okna są nieważne.



Rysunek 13.12. 8-rejestrowe okno SPARC tworzące kolisty stos

W architekturze SPARC nie jest zwykle konieczne zapisywanie ani przywracanie rejestrów w przypadku wywołania procedury. Kompilator jest uproszczony, ponieważ do jego zadań należy jedynie efektywne przydzielanie rejestrów lokalnych procedurze i nie musi on się troszczyć o alokację rejestrów między procedurami.

Lista rozkazów

Rozkazy używane w architekturze SPARC zostały wymienione w tabeli 13.14. Większość z nich odnosi się wyłącznie do argumentów znajdujących się w rejestrach. Rozkazy typu rejestr-rejestr mają trzy argumenty i mogą być wyrażone w następującej postaci:

$R_d \leftarrow R_{S1}$ op S2

gdzie R_d i R_{S1} są odniesieniami do rejestrów, a S2 może być albo odniesieniem do rejestru, albo do 13-bitowego argumentu natychmiastowego. Zero rejestrowe (R_0) jest ustawione sprzętowo wartościami 0. Taka postać jest dobrze dostosowana do typowych programów, w których mamy do czynienia z dużym udziałem lokalnych skalarów i stałych.

Tabela 13.14. Lista rozkazów SPARC

Operacja	Opis	Operacja	Opis
	Rozkazy ładowania i zapisu		**Rozkazy arytmetyczne**
LDSB	Ładowanie bajta ze znakiem	ADD	Dodawanie
LDSH	Ładowanie półsłowa ze znakiem	ADDCC	Dodawanie, ustawianie icc
LDUB	Ładowanie bajta bez znaku	ADDX	Dodawanie z przeniesieniem
LDUH	Ładowanie półsłowa bez znaku	ADDXCC	Dodawanie z przeniesieniem, ustawianie icc
LD	Ładowanie słowa	SUB	Odejmowanie
LDD	Ładowanie podwójnego słowa	SUBCC	Odejmowanie, ustawianie icc
STB	Zapisywanie bajta	SUBX	Odejmowanie z przeniesieniem
STH	Zapisywanie półsłowa	SUBXCC	Odejmowanie z przeniesieniem, ustawianie icc
STD	Zapisywanie słowa	MULSCC	Mnożenie, ustawianie icc
STDD	Zapisywanie podwójnego słowa		**Rozkazy skoku i rozgałęzienia**
	Rozkazy przesunięcia	BCC	Rozgałęzienie na podstawie warunku
SLL	Logiczne przesunięcie w lewo	FBCC	Rozgałęzienie na podstawie warunku zmiennopozycyjnego
SRL	Logiczne przesunięcie w prawo	CBCC	Rozgałęzienie na podstawie warunku koprocesora
SRA	Arytmetyczne przesunięcie w prawo	CALL	Wywołanie procedury
	Rozkazy boolowskie	JMPL	Skok i łącze
AND	AND	TCC	Pułapka na podstawie warunku
ANDCC	AND, ustawianie icc	SAVE	Przesunięcie okna rejestru do przodu
ANDN	NAND	RESTORE	Przesunięcie okien wstecz
ANDNCC	NAND, ustawianie icc	RETT	Powrót z pułapki
OR	OR		**Rozkazy różne**
ORCC	OR, ustawianie icc	SETHI	Ustawianie 22 bitów wyższego rzędu
ORN	NOR	UNIMP	Rozkaz nieimplementowany (pułapka)
ORNCC	NOR, ustawianie icc	RD	Odczytanie rejestru specjalnego
XOR	XOR	WR	Zapisanie w rejestrze specjalnym
XORCC	XOR, ustawianie icc	IFLUSH	Opróżnianie pamięci podręcznej rozkazów
XNOR	Wyłączne NOR		
XNORCC	Wyłączne NOR, ustawianie icc		

Dostępne operacje jednostki arytmetyczno-logicznej mogą być pogrupowane następująco:

- Dodawanie liczb całkowitych (z przeniesieniem lub bez).
- Odejmowanie liczb całkowitych (z przeniesieniem lub bez).
- Operacje boolowskie AND, OR, XOR na poziomie bitowym i ich negacje.
- Przesunięcia logiczne w lewo i w prawo oraz arytmetyczne w prawo.

Wszystkie te rozkazy – z wyjątkiem przesunięć – mogą opcjonalnie ustawiać cztery kody warunkowe (ZERO, NEGATIVE, OVERFLOW, CARRY). Liczby całkowite ze znakiem są reprezentowane w postaci 32-bitowego uzupełnienia do dwóch.

Tylko proste rozkazy ładowania i zapisywania odwołują się do pamięci. Istnieją odrębne rozkazy ładowania i zapisu słowa (32-bitowego), podwójnego słowa, półsłowa i bajta. W dwóch ostatnich przypadkach istnieją rozkazy ładowania tych wielkości jako liczb ze znakiem lub bez znaku. Liczby ze znakiem mają poszerzony znak w celu wypełnienia 32-bitowego rejestru docelowego. Liczby bez znaku są uzupełniane zerami.

Jedynym dostępnym trybem adresowania – poza rejestrowym – jest tryb z przesunięciem. Oznacza to, że efektywny adres argumentu składa się z przesunięcia w stosunku do adresu zawartego w rejestrze:

$$EA = (R_{S1}) + S2$$

$$\text{lub } EA = (R_{S1}) + (R_{S2})$$

zależnie od tego, czy drugi argument jest natychmiastowy, czy też znajduje się w rejestrze. W celu realizacji ładowania lub zapisu do cyklu rozkazu jest dodawany dodatkowy etap. Podczas etapu drugiego za pomocą jednostki arytmetyczno-logicznej jest obliczany adres; ładowanie lub zapisywanie następuje w trzecim etapie. Ten prosty tryb adresowania jest całkiem wszechstronny i może służyć do syntetyzowania innych trybów adresowania, co zostało pokazane w tabeli 13.15.

Tabela 13.15. Syntetyzowanie pozostałych trybów adresowania za pomocą trybów adresowania SPARC

Tryb	Algorytm	Równoważnik SPARC	Typ instrukcji
Natychmiastowy	Argument = A	S2	Rejester-rejestr
Bezpośredni	EA = A	$R_0 + S2$	Ładowanie, zapisywanie
Rejestrowy	EA = R	$R_{S1} + R_{S2}$	Rejestr-rejestr
Rejestrowy pośredni	EA = (R)	$R_{S1} + 0$	Ładowanie, zapisywanie
Przemieszczeniowy	EA = (R) + A	$R_{S1} + S2$	Ładowanie, zapisywanie

Pouczające jest porównanie możliwości adresowania SPARC i MIPS. W MIPS wykorzystuje się wyrównanie (*offset*) 16-bitowe; w SPARC jest ono 13-bitowe. Z drugiej strony, w MIPS nie jest możliwe budowanie adresu na podstawie zawartości dwóch rejestrów.

Format rozkazu

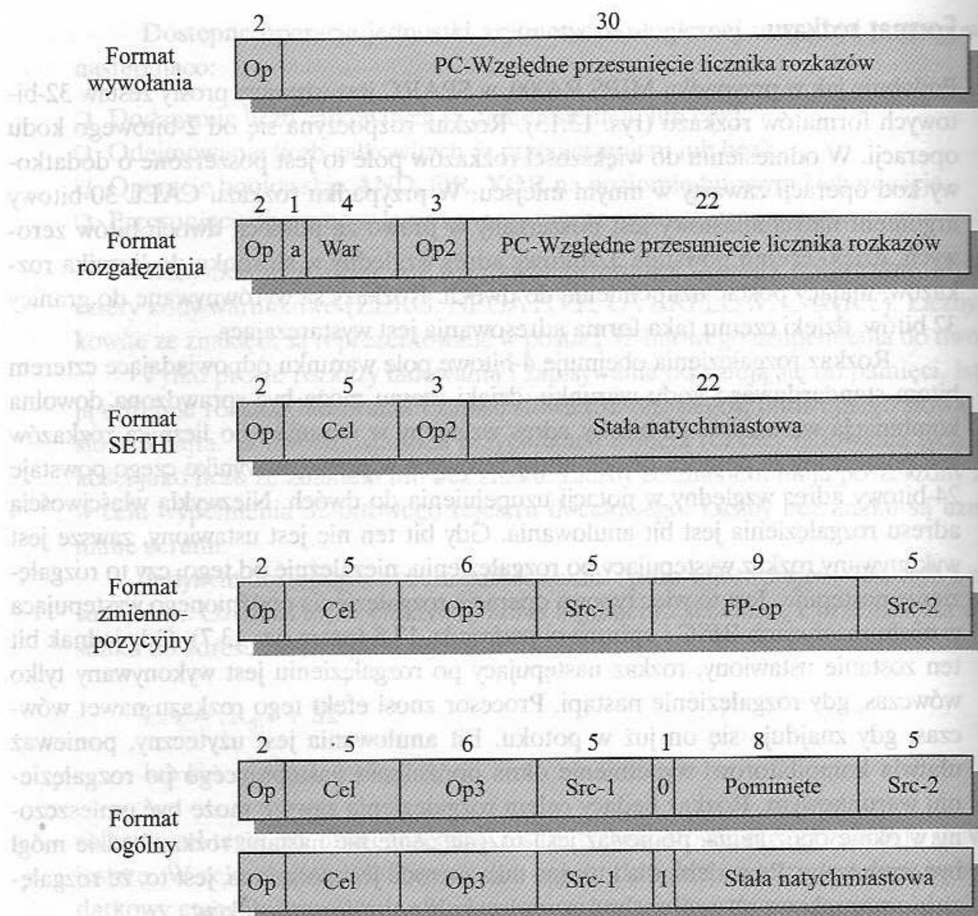
Podobnie jak w przypadku MIPS R4000, w SPARC jest używany prosty zestaw 32-bitowych formatów rozkazu (rys. 13.13). Rozkaz rozpoczyna się od 2-bitowego kodu operacji. W odniesieniu do większości rozkazów pole to jest poszerzone o dodatkowy kod operacji zawarty w innym miejscu. W przypadku rozkazu CALL 30-bitowy argument natychmiastowy jest poszerzany w prawo za pomocą dwóch bitów zerowych, dzięki czemu powstaje 32-bitowy adres względny w stosunku do licznika rozkazów, mający postać uzupełnienia do dwóch. Rozkazy są wyrównywane do granicy 32 bitów, dzięki czemu taka forma adresowania jest wystarczająca.

Rozkaz rozgałęzienia obejmuje 4-bitowe pole warunku odpowiadające czterem bitom standardowego kodu warunku, dzięki czemu może być sprawdzona dowolna kombinacja warunków. 22-bitowy adres względny w stosunku do licznika rozkazów jest poszerzany za pomocą dwóch bitów zerowych w prawo, w wyniku czego powstaje 24-bitowy adres względny w notacji uzupełnienia do dwóch. Niezwykłą właściwością adresu rozgałęzienia jest bit anulowania. Gdy bit ten nie jest ustawiony, zawsze jest wykonywany rozkaz występujący po rozgałęzieniu, niezależnie od tego, czy to rozgałęzienie następuje. Jest to więc typowa operacja rozgałęzienia opóźnionego występująca w wielu maszynach RISC i opisana w podrozdz. 13.5 (patrz rys. 13.7). Gdy jednak bit ten zostanie ustawiony, rozkaz następujący po rozgałęzieniu jest wykonywany tylko wówczas, gdy rozgałęzienie nastąpi. Procesor znosi efekt tego rozkazu nawet wówczas, gdy znajduje się on już w potoku. Bit anulowania jest użyteczny, ponieważ ułatwia kompilatorowi wypełnienie okna opóźnienia następującego po rozgałęzieniu warunkowym. Rozkaz będący celem rozgałęzienia zawsze może być umieszczony w oknie opóźnienia, ponieważ jeśli rozgałęzienie nie nastąpi, rozkaz będzie mógł być anulowany. Powodem, dla którego taka metoda jest pożądana, jest to, że rozgałęzienia warunkowe na ogół zachodzą w więcej niż połowie przypadków.

Rozkaz SETHI jest rozkazem specjalnym, służącym do ładowania lub zapisywania wartości 32-bitowych. Właściwość ta jest potrzebna do ładowania i zapisywania adresów oraz dużych stałych. Rozkaz ten ustawia 22 bity wyższego rzędu rejestru za pomocą 22-bitowego argumentu i zeruje 10 bitów niższego rzędu. Natychmiastowa stała o długości do 13 bitów może występować w jednym z formatów ogólnych, a tego rodzaju rozkaz może służyć do wypełnienia pozostałych 10 bitów rejestru. Rozkaz ładowania lub zapisu może być również użyty do uzyskiwania trybu adresowania bezpośredniego. W celu załadowania wartości z lokacji K w pamięci moglibyśmy użyć następujących rozkazów SPARC:

```
sethi   %hi(K), %r8            ;ładowanie 22 bitów wyższego rzędu
                               ;adresu z lokacji K do rejestru r8
ld      [%r8 + %lo(K)], %r8    ;ładowanie zawartości lokacji
                               ;K do r8
```

Makra %hi i %lo zostały użyte do zdefiniowania argumentów natychmiastowych, składających się z odpowiednich bitów adresu danej lokacji. Takie użycie SETHI przypomina stosowanie rozkazu LUI w MIPS (tabela 13.12).



Rysunek 13.13. Formaty instrukcji SPARC

W operacjach zmiennopozycyjnych jest używany format zmiennopozycyjny. Wyznaczane są dwa rejestry źródłowe i jeden docelowy.

Wszystkie pozostałe operacje, łącznie z ładowaniem, zapisem, operacjami arytmetycznymi i logicznymi korzystają z jednego z dwóch formatów pokazanych na rys. 13.13. W jednym z formatów są używane dwa rejestry źródłowe i jeden docelowy, podczas gdy w pozostałym wykorzystuje się jeden rejestr źródłowy, jeden natychmiastowy argument 13-bitowy i jeden rejestr docelowy.

13.8. Porównanie architektur RISC i CISC

Przez wiele lat główną tendencją objawiającą się w rozwoju architektury i organizacji komputerów było zwiększanie złożoności procesora: więcej rozkazów, więcej trybów adresowania, więcej wyspecjalizowanych rejestrów i tak dalej. Rozwój architektury

RISC oznaczał całkowite odejście od filozofii leżącej u podstaw tej tendencji. Naturalnie pojawienie się systemów RISC i publikacji wysławiających cnoty RISC spowodowało reakcję ze strony tych, którzy byli zaangażowani w projektowanie architektur CISC.

Prace nad oceną zalet rozwiązania RISC można podzielić na dwie kategorie:

- **Ilościowe.** Dążenie do porównania rozmiaru programów i szybkości ich realizacji na maszynach RISC i CISC wykonanych z wykorzystaniem porównywalnej technologii.
- **Jakościowe.** Badanie zagadnień, takich jak wspieranie języków wysokiego poziomu i optymalne wykorzystywanie możliwości VLSI.

Większość prac nad oceną ilościową została wykonana przez zespoły pracujące nad systemami RISC [PATT82b, HEAT84, PATT84]; prace te w znacznej większości faworyzowały rozwiązanie RISC. Inni badali to zagadnienie i pozostali nieprzekonani [COLW85a, FLYN87, DAVI87]. Istnieje kilka problemów z przeprowadzeniem takich porównań [SERL86]:

- Nie ma pary maszyn RISC i CISC, które są porównywalne pod względem całkowitych kosztów, poziomu technologii, stopnia scalenia, złożoności kompilatora, wsparcia systemu operacyjnego itd.
- Nie istnieje rozstrzygający zbiór testów programów. Wydajność zależy od programu.
- Trudne jest oddzielenie wpływów sprzętu od wyników związanych z biegłością twórców kompilatorów.
- Większość analiz porównawczych dotyczących architektury RISC była wykonana raczej na maszynach-zabawkach niż na wyrobach komercyjnych. Ponadto większość maszyn dostępnych na rynku i reklamowanych jako RISC ma zarówno cechy architektury RISC, jak i CISC. Dlatego uczciwe porównanie z komercyjną maszyną o czystej architekturze CISC (np. VAX, Pentium) jest trudne.

Ocena jakościowa jest subiektywna niemal z definicji. Kilku badaczy zwróciło uwagę na prace [COLW85a, WALL85], jednak w najlepszym razie ich rezultaty są niejednoznaczne; są też kwestionowane [PATT85b] i oczywiście również bronione [COLW85b].

W ciągu ostatnich lat polemika „RISC versus CISC" w znacznej mierze zanikła. Stało się tak z powodu stopniowego przenikania się technologii. W miarę wzrostu gęstości upakowania mikroukładów i szybkości działania samego sprzętu systemy RISC stały się bardziej złożone. Jednocześnie w ramach dążenia do osiągnięcia maksymalnej wydajności, projektowanie architektur CISC skupiło się na zagadnieniach tradycyjnie związanych z RISC, takich jak wzrost liczby rejestrów ogólnego przeznaczenia i zwiększenie nacisku na projektowanie potoku rozkazów.

13.9. Polecana literatura

Podręczniki zawierające dobre ujęcie koncepcji RISC to [WARD90], [PATT98] i [HENN96].

W [KANE92] szczegółowo przedstawiono komercyjne komputery MIPS. W [MIRA92] znajduje się dobry przegląd własności MIPS R4000. W [BASH91] przeanalizowano ewolucję od potoku R3000 do superpotoku R4000. SPARC przedstawiono dość szczegółowo w [DEWA90].

BASH91 Bashteen A., Lui I., Mullan J.: „A Superpipeline Approach to the MIPS Architecture". *Proceedings, COMPCON Spring '91*, February 1991.

DEWA90 Dewar R., Smosna M.: *Microprocessors: A Programmer's View*. New York, McGraw-Hill, 1990.

HENN96 Hennessy J., Patterson D.: *Computer Architecture: A Quantitative Approach*. San Mateo, Morgan Kaufmann, 1996.

KANE92 Kane G., Heinrich J.: *MIPS RISC Architecture*. Englewood Cliffs, Prentice Hall, 1992.

MIRA92 Mirapuri S., Woodacre M., Vasseghi N.: „The MIPS R4000 Processor". *IEEE Micro*, April 1992.

PATT98 Patterson D., Hennessy J.: *Computer Organization and Design: The Hardware/Software Interface*. San Mateo, Morgan Kaufmann, 1998.

WARD90 Ward S., Halstead R.: *Computation Structures*. Cambridge, MIT Press, 1990.

13.10. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Język wysokiego poziomu – *high level language (HLL)*
Komputer o złożonej liście rozkazów (CISC) – *complex instruction set computer*
Komputer o zredukowanej liście rozkazów (RISC) – *reduced instruction set computer*
Ładowanie opóźnione – *delayed load*
Okno rejestrów – *register window*
Rozgałęzienie opóźnione – *delayed branch*
SPARC – *scalable procesor architecture*
Tablica rejestrów – *register file*

Pytania kontrolne

13.1. Jakie są typowe właściwości wyróżniające organizację RISC?

13.2. Krótko objaśnij dwa podstawowe rozwiązania służące do minimalizowania operacji rejestr-pamięć w komputerach RISC.

13.3. Jeśli do operowania lokalnymi zmiennymi procedur zagnieżdżonych służy cykliczny bufor rejestrowy, opisz dwa rozwiązania operowania zmiennymi globalnymi.

13.4. Jakie są typowe właściwości architektury listy rozkazów RISC?

13.5. Co to jest rozgałęzienie opóźnione?

Problemy do rozwiązania

13.1. Rozważ przebieg wywołań i powrotów przedstawiony na rys. 4.16. Ile przepełnień i niedomiarów (z których każde powoduje operację zapisywania i odnawiania zawartości rejestru) nastąpi przy rozmiarze okna:

(a) 5;

(b) 8;

(c) 16.

13.2. Omawiając rys. 13.2, stwierdziliśmy, że tylko dwie pierwsze części okna są zapisywane i odtwarzane. Dlaczego nie jest konieczne zapisywanie zawartości rejestrów tymczasowych?

13.3. Chcemy wyznaczyć czas wykonywania danego programu, stosując różne schematy przetwarzania potokowego rozważane w podrozdz. 13.5. Oznaczmy:

N – liczba wykonanych rozkazów;

D – liczba dostępów do pamięci;

J – liczba rozkazów skoku.

W przypadku prostego schematu sekwencyjnego (rys. 13.6a) czas wykonania wynosi $2N + D$ faz. Wyprowadź wzory dla przetwarzania potokowego 2-, 3- i 4-etapowego.

13.4. Rozważ następujący fragment programu w języku wysokiego poziomu:

```
for I in 1 . . . 100 loop
    S ← S + Q(I).VAL
end loop;
```

Załóż, że Q jest tablicą rekordów 32-bajtowych i że pole VAL znajduje się w pierwszych 4 bajtach każdego rekordu. Stosując kod procesora 80x86, możemy skompilować ten fragment programu następująco:

```
      MOV    ECX, 1        ;użyj rejestru ECX do przechowania 1
LP:   IMUL   EAX, ECX, 32  ;znajdź przesunięcie w EAX
      MOV    EBX, Q[EAX]   ;ładuj pole VAL
      ADD    S, EBX        ;dodaj do S
      INC    ECX           ;zwiększ stan I
      CMP    ECX, 100      ;sprawdź w stosunku do granicy
      JNE    LP            ;pętla aż do I = 100.
```

Program ten wykorzystuje rozkaz IMUL, powodujący mnożenie drugiego argumentu przez wartość natychmiastową w trzecim argumencie i umieszczenie wyniku w pierwszym argumencie (patrz problem 10.13). Zwolennik architektury RISC chciałby zademonstrować, że inteligentny kompilator może wyeliminować niepotrzebnie złożone rozkazy, takie jak IMUL. Zademonstruj to, przerabiając powyższy program procesora 80x86 bez użycia rozkazu IMUL.

13.5. Rozważ następującą pętlę:

```
S : = 0;
for K : = 1 to 100 do
        S : = S – K;
```

Prosta translacja tego na ogólny język asemblerowy mogłaby wyglądać następująco:

```
    LD    R1, 0          ;zatrzymaj wartość S w R1
    LD    R2, 1          ;zatrzymaj wartość K w R2
LP  SUB   R1, R1, R2     ;S := S - K
    BEQ   R2, 100, EXIT  ;wykonywane, jeśli K = 100
    ADD   R2, R2, 1      ;w przeciwnym przypadku inkrementacja K
    JMP   LP             ;powrót do początku pętli
```

Kompilator maszyny RISC wprowadzi do tego kodu okna opóźnień, dzięki czemu procesor będzie mógł wykorzystać mechanizm rozgałęzienia opóźnionego. Rozkaz JMP nie stanowi problemu, ponieważ po nim zawsze następuje rozkaz SUB; możemy po prostu umieścić kopię rozkazu SUB w oknie opóźnienia po JMP. Pewną trudność stanowi BEQ. Nie możemy pozostawić tego kodu w istniejącej postaci, ponieważ rozkaz ADD byłby wykonany o jeden raz za dużo. Potrzebny jest więc rozkaz NOOP. Podaj kod wynikowy.

13.6. Dodaj do tabeli 13.8 zapisy dotyczące procesorów:
(a) Pentium II;
(b) PowerPC.

13.7. W wielu przypadkach wspólne rozkazy maszynowe, nie wymienione w ramach listy rozkazów MIPS, mogą być syntetyzowane w postaci pojedynczego rozkazu MIPS. Pokaż tę możliwość dla:
(a) przeniesienia z rejestru do rejestru;
(b) inkrementacji i dekrementacji;
(c) uzupełnienia;
(d) zanegowania;
(e) zerowania.

13.8. Pewna implementacja SPARC ma okna *K*-rejestrowe. Jaka jest liczba rejestrów fizycznych (*N*)?

13.9. W architekturze SPARC nie ma wielu rozkazów używanych powszechnie w komputerach CISC. Niektóre z nich można łatwo symulować za pomocą albo rejestru R0, który jest zawsze wyzerowany, albo argumentu w postaci stałej. Takie symulowane rozkazy są nazywane *pseudorozkazami* i są rozpoznawane przez kompilator SPARC. Pokaż, jak można symulować następujące pseudorozkazy, każdy z nich za pomocą jednego rozkazu SPARC. We wszystkich przypadkach src i dst odnoszą się do rejestrów. *Wskazówka*: zapisywanie A w R0 nie daje żadnego efektu.

(a) MOV src, dst	(d) NOT dst	(g) DEC dst
(b) COMPARE src1, src2	(e) NEG dst	(h) CLR dst
(c) TEST src1	(f) INC dst	(i) NOOP

13.10. Rozważ następujący fragment kodu:

```
if K > 10
      L : = K + 1
else
      L : = K - 1;
```

Prosta translacja tej instrukcji na język asemblerowy SPARC mogłaby mieć postać następującą:

```
      sethi  %hi(K), %r8              ;ładowanie 22 bitów wysokiego rzędu
                                      ;adresu
                                      ;lokacji K do rejestru r8
      ld     [%r8 + %lo(K)], %r8      ;ładowanie zawartości lokacji K do r8
      cmp    %r8, 10                  ;porównanie zawartości r8 z 10
      ble    L1                       ;rozgałęzienie, jeśli (r8) ≤ 10
      noop
      sethi  %hi(K), %r9
      ld     [%r9 + %lo(K)], %r9      ;ładowanie zawartości lokacji K do r9
      inc    %r9                      ;dodawanie 1 do (r9)
      sethi  %hi(L), %r10
      st     %r9, [%r10 + %lo(L)]     ;zapisywanie (r9) w lokacji L
      b      L2
      noop
L1:   sethi  %hi(K), %r11
      ld     [%r11 + %lo(K)], %r12    ;ładowanie zawartości lokacji K do r12
      dec    %r12                     ;odejmowanie 1 od (r12)
      sethi  %hi(L), %r13
      st     %r12, [%r13 + %lo(L)]    ;zapisywanie (r12) w lokacji L
L2:
```

Kod ten zawiera rozkaz `noop` po każdym rozkazie rozgałęzienia, aby umożliwić operację rozgałęziania opóźnionego.

(a) Standardowe optymalizacje oparte na kompilatorze, które nie mają nic wspólnego z komputerami RISC, umożliwiają dokonanie dwóch przekształceń powyższego kodu. Zauważ, że dwie spośród operacji ładowania nie będą konieczne i że dwie operacje zapisu będą mogły być połączone, jeśli zapis zostanie przeniesiony do innego miejsca kodu. Pokaż ten program po wprowadzeniu tych dwóch zmian.

(b) Jest teraz możliwe dokonanie pewnych optymalizacji właściwych dla SPARC. Operacja `noop` po operacji `ble` może być zastąpiona przez przeniesienie innego rozkazu do okna opóźnienia oraz ustawienie bitu anulowania w rozkazie `ble` (co można wyrazić jako `ble,a L1`). Pokaż program po tych zmianach.

(c) Dwa rozkazy są teraz niepotrzebne. Usuń je i pokaż program wynikowy.

daj kod wynikowy.

13.6 Dodaj do tabeli 13.x zapisy dotyczące procesorów:

(a) Pentium II

(b) PowerPC

13.7 W wielu przypadkach wspólne rozkazy maszynowe ...

(b) inkrementacji i dekrementacji

rozkazu SPARC. We wszystkich przypadkach src i dst odnoszą się do rejestrów. *Wskazówka*: zapisywanie A w R0 nie daje żadnego efektu.

(a) MOV src, dst	(d) NOT dst	(g) DEC dst
(b) COMPARE src1, src2	(e) NEG dst	(h) CLR dst
(c) TEST src1	(f) INC dst	(i) NOOP

13.10 Rozważ następujący fragment kodu:

```
if K > 10
    L := K + 1
else
    L := K - 1;
```

Prosta translacja tej instrukcji na język asemblerowy SPARC mogłaby mieć postać następującą:

Rozdział 14

Paralelizm na poziomie rozkazu i procesory superskalarne

14.1. Przegląd

- Rozwiązanie superskalarne a przetwarzanie superpotokowe
- Ograniczenia

14.2. Problemy projektowania

- Paralelizm na poziomie rozkazu i paralelizm na poziomie maszyny
- Strategia wydawania rozkazów
- Przemianowanie rejestrów
- Paralelizm maszynowy
- Przewidywanie rozgałęzień
- Superskalarne wykonywanie programów
- Implementacja superskalarna

14.3. Pentium 4

- Część czołowa
- Układy logiczne wykonywania rozkazów poza kolejnością
- Jednostki wykonywania: całkowitoliczbowa i zmiennopozycyjna

14.4. PowerPC

- PowerPC 601
- Przetwarzanie rozgałęzień
- PowerPC 620

14.5. Polecana literatura

14.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

- Podstawowe terminy i ich angielskie odpowiedniki
- Pytania kontrolne
- Problemy do rozwiązania

Podstawowe spostrzeżenia

- Procesor superskalarny charakteryzuje się użyciem wielu niezależnych potoków rozkazów. Każdy potok składa się z wielu etapów, dzięki czemu jest w stanie operować jednocześnie wieloma rozkazami. Wiele potoków oznacza nowy poziom paralelizmu, umożliwiając jednoczesne przetwarzanie wielu strumieni rozkazów. W procesorze superskalarnym jest stosowane rozwiązanie określane jako paralelizm na poziomie rozkazu, co odnosi się do stopnia, w jakim rozkazy programu mogą być wykonywane równolegle.
- Procesor superskalarny pobiera zwykle jednocześnie wiele rozkazów, po czym próbuje znaleźć bliskie i niezależne od siebie rozkazy, które mogą być wykonywane równolegle. Jeśli dane wejściowe jednego rozkazu zależą od danych wyjściowych poprzedniego rozkazu, to wykonywanie tego rozkazu nie może być ukończone przed wykonaniem poprzedniego. Gdy tego rodzaju zależności zostaną zidentyfikowane, procesor może wydawać rozkazy i kończyć ich wykonywanie w kolejności, która różni się od wynikającej z początkowego kodu maszynowego.
- Procesor może eliminować pewne niekonieczne zależności przez zastosowanie dodatkowych rejestrów i przemianowywanie odniesień do rejestrów w kodzie początkowym.
- Podczas gdy „czyste" procesory RISC często posługują się rozgałęzieniami opóźnionymi w celu maksymalizacji wykorzystania potoku rozkazów, metoda ta nie jest już tak odpowiednia dla maszyn superskalarnych. Zamiast tego, w celu zwiększenia efektywności stosuje się w nich tradycyjne sposoby przewidywania rozgałęzień.

W procesorze o architekturze superskalarnej zwykłe rozkazy, na przykład dotyczące arytmetyki liczb całkowitych lub zmiennopozycyjnych, ładowania, zapisu i rozgałęzienia warunkowego, mogą być równocześnie inicjowane i niezależnie wykonywane. Rozwiązanie takie stwarza wiele złożonych problemów projektowych związanych z potokowym przetwarzaniem rozkazów.

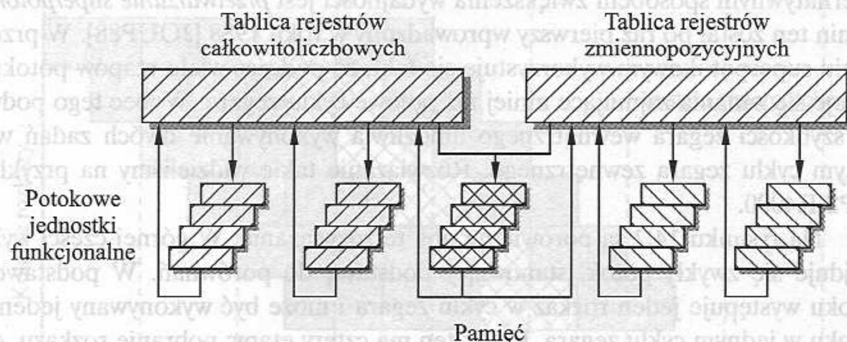
Projekty superskalarne pojawiły się na scenie, depcząc po piętach architekturze RISC. Chociaż architektura o uproszczonej liście rozkazów maszyn RISC sprzyja metodom superskalarnym, to rozwiązania superskalarne mogą być stosowane zarówno w architekturze RISC, jak i CISC. Jednak praktycznie wszystkie wdrożenia superskalarne zostały oparte na architekturze RISC.

Podczas gdy okres dojrzewania – między początkiem badań nad RISC w postaci maszyn IBM 801 i Berkeley RISC I a ukazaniem się komercyjnych maszyn RISC – trwał siedem lub osiem lat, pierwsze maszyny superskalarne ukazały się rok lub dwa po wymyśleniu terminu „superskalarny". Rozwiązanie superskalarne stało się obecnie standardową metodą implementacji mikroprocesorów o wysokiej wydajności.

Rozpoczniemy ten rozdział od przeglądu podejścia superskalarnego, przeciwstawiając je przetwarzaniu superpotokowemu. Następnie przedstawimy główne zagadnienia projektowe związane z rozwiązaniem superskalarnym, a na zakończenie zostanie przedstawionych kilka ważnych przykładów architektury superskalarnej.

14.1. Przegląd

Termin *superskalarny*, użyty po raz pierwszy w roku 1987 [AGER87], odnosi się do maszyny, która została zaprojektowana pod kątem zwiększenia wydajności wykonywania rozkazów skalarnych. W większości zastosowań główna część operacji jest prowadzona na wielkościach skalarnych. Wobec tego rozwiązanie superskalarne reprezentuje następny krok w ewolucji wysoko wydajnych procesorów o ogólnym przeznaczeniu.



Rysunek 14.1. Ogólna organizacja superskalarna [COME95]

Istotą rozwiązania superskalarnego jest zdolność do niezależnego wykonywania rozkazów w różnych potokach. Koncepcja ta może być pogłębiona, aby było możliwe wykonywanie rozkazów w kolejności odmiennej niż wynikająca z programu. Na rysunku 14.1 została pokazana ogólna postać przykładowego rozwiązania superskalarnego. Widocznych jest wiele jednostek funkcjonalnych, z których każda została implementowana jako potok, co umożliwia równoległe wykonywanie kilku rozkazów. W pokazanym przykładzie jest możliwe jednoczesne wykonywanie dwóch operacji całkowitoliczbowych, dwóch zmiennopozycyjnych i jednej pamięciowej (ładowania lub zapisu).

Wielu badaczy analizowało procesory zbliżone do superskalarnych, a wyniki ich badań wskazują na możliwość pewnej poprawy wydajności. Stwierdzony stopień poprawy wydajności jest przedstawiony w tabeli 14.1. Różnice rezultatów wynikają zarówno z różnic sprzętowych symulowanych maszyn, jak i z symulowanych zastosowań.

Tabela 14.1. Doniesienia o przyspieszeniu pracy komputerów dzięki rozwiązaniu superskalarnemu

Źródło	Przyspieszenie
[TJAD70]	1,8
[KUCK72]	8
[WEIS84]	1,58
[ACOS86]	2,7
[SOHI90]	1,8
[SMIT89]	2,3
[JOUP89]	2,2
[LEE91]	7

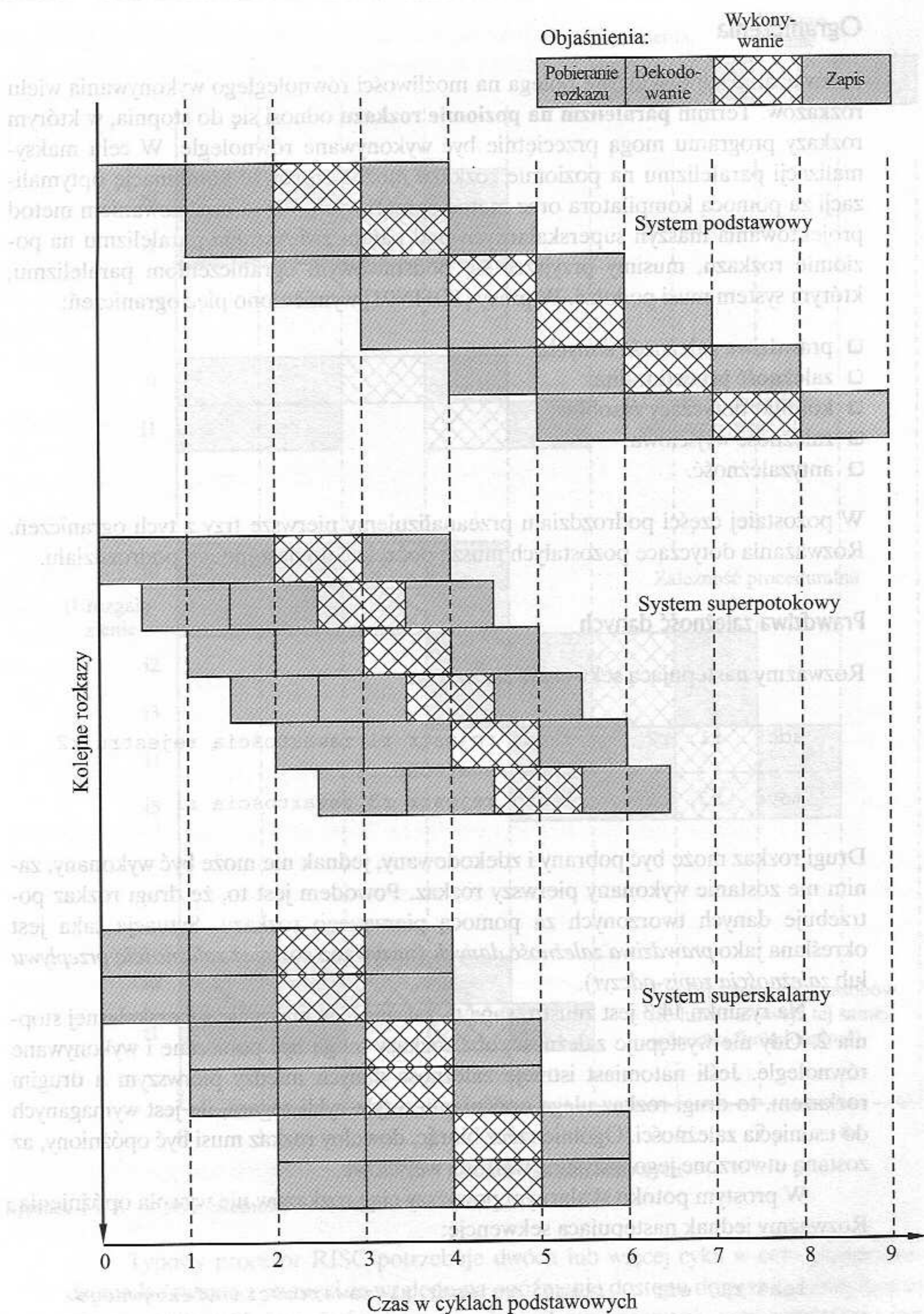
Rozwiązanie superskalarne a przetwarzanie superpotokowe

Alternatywnym sposobem zwiększenia wydajności jest *przetwarzanie superpotokowe*; termin ten został po raz pierwszy wprowadzony w roku 1988 [JOUP88]. W przetwarzaniu superpotokowym wykorzystuje się fakt, że podczas wielu etapów potoku wykonuje się zadania zajmujące mniej niż połowę cyklu zegara. Wobec tego podwojenie szybkości zegara wewnętrznego umożliwia wykonywanie dwóch zadań w tym samym cyklu zegara zewnętrznego. Rozwiązanie takie widzieliśmy na przykładzie MIPS R4000.

Na rysunku 14.2 są porównane oba te rozwiązania. W górnej części wykresu znajduje się zwykły potok, stanowiący podstawę do porównań. W podstawowym potoku występuje jeden rozkaz w cyklu zegara i może być wykonywany jeden etap potoku w jednym cyklu zegara. Potok ten ma cztery etapy: pobranie rozkazu, dekodowanie operacji, wykonanie operacji i zapisanie wyniku. Zauważmy, że chociaż kilka rozkazów jest wykonywanych równocześnie, w określonej chwili tylko jeden rozkaz znajduje się na etapie wykonywania.

Następna część wykresu ilustruje rozwiązanie superpotokowe, umożliwiające realizowanie dwóch etapów potoku w każdym cyklu zegara. Alternatywnym sposobem widzenia tego rozwiązania jest rozdzielenie funkcji wykonywanych na każdym etapie na dwie nienakładające się części, z których każda może być wykonywana w połowie cyklu zegara. Rozwiązanie superpotokowe tego rodzaju jest określane jako rozwiązanie stopnia 2. Wreszcie najniższa część wykresu ilustruje rozwiązanie superskalarne, umożliwiające równoległe wykonywanie dwóch przypadków każdego etapu. Możliwe są oczywiście superpotoki i rozwiązania superskalarne wyższego stopnia.

Zarówno w rozwiązaniu superpotokowym, jak i superskalarnym przedstawionych na rys. 14.2 występuje taka sama liczba rozkazów wykonywanych jednocześnie w stanie ustalonym. Procesor superpotokowy przegrywa z procesorem superskalarnym na początku programu i przy każdym punkcie docelowym rozgałęzienia.



Rysunek 14.2. Porównanie rozwiązań superskalarnego i superpotokowego

Ograniczenia

Rozwiązanie superskalarne polega na możliwości równoległego wykonywania wielu rozkazów. Termin **paralelizm na poziomie rozkazu** odnosi się do stopnia, w którym rozkazy programu mogą przeciętnie być wykonywane równolegle. W celu maksymalizacji paralelizmu na poziomie rozkazu można stosować kombinację optymalizacji za pomocą kompilatora oraz metod sprzętowych. Przed analizowaniem metod projektowania maszyn superskalarnych pod kątem zwiększania paralelizmu na poziomie rozkazu, musimy przyjrzeć się podstawowym ograniczeniom paralelizmu, którym system musi podołać. W pracy [JOHN91] wymieniono pięć ograniczeń:

- prawdziwa zależność danych;
- zależność proceduralna;
- konflikt dotyczący zasobów;
- zależność wyjściowa;
- antyzależność.

W pozostałej części podrozdziału przeanalizujemy pierwsze trzy z tych ograniczeń. Rozważania dotyczące pozostałych muszą poczekać do następnego podrozdziału.

Prawdziwa zależność danych

Rozważmy następującą sekwencję rozkazów:

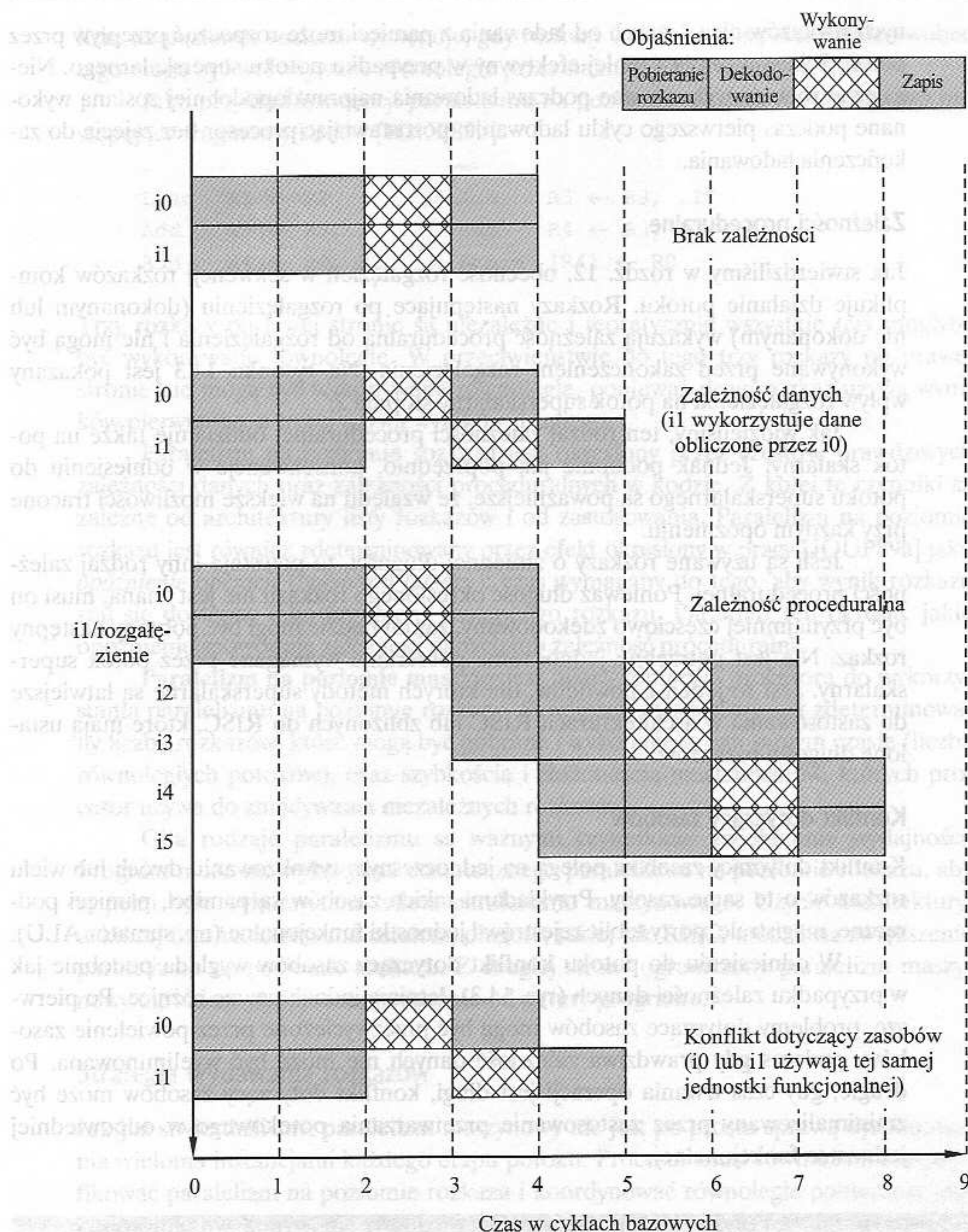
```
add   r1, r2    ;ładuj rejestr r1 zawartością rejestru r2
                  plus zawartość r1
move  r3, r1    ;ładuj rejestr r3 zawartością r1
```

Drugi rozkaz może być pobrany i zdekodowany, jednak nie może być wykonany, zanim nie zostanie wykonany pierwszy rozkaz. Powodem jest to, że drugi rozkaz potrzebuje danych tworzonych za pomocą pierwszego rozkazu. Sytuacja taka jest określana jako *prawdziwa zależność danych* (nazywana również *zależnością przepływu* lub *zależnością zapis-odczyt*).

Na rysunku 14.3 jest zilustrowana ta zależność w maszynie superskalarnej stopnia 2. Gdy nie występuje zależność, oba rozkazy mogą być pobierane i wykonywane równolegle. Jeśli natomiast istnieje zależność danych między pierwszym a drugim rozkazem, to drugi rozkaz ulega opóźnieniu o tyle cykli zegara, ile jest wymaganych do usunięcia zależności. Ogólnie rzecz biorąc, dowolny rozkaz musi być opóźniony, aż zostaną utworzone jego wszystkie wartości wejściowe.

W prostym potoku skalarnym powyższy ciąg rozkazów nie wywoła opóźnienia. Rozważmy jednak następującą sekwencję:

```
load r1, eff   ;ładuj rejestr r1 zawartością efektywnego
                adresu pamięci eff
move r3, r1    ;ładuj rejestr r3 zawartością r1
```



Rysunek 14.3. Wpływ zależności

Typowy procesor RISC potrzebuje dwóch lub więcej cykli w celu przeprowadzenia ładowania z pamięci ze względu na opóźnienie dostępu do pamięci znajdującej się poza mikroukładem. Jednym ze sposobów skompensowania tego opóźnienia jest zmiana kolejności rozkazów przez kompilator, dzięki czemu jeden lub więcej kolej-

nych rozkazów niezależnych od ładowania z pamięci może rozpocząć przepływ przez potok. Schemat ten jest mniej efektywny w przypadku potoku superskalarnego. Niezależne rozkazy wykonywane podczas ładowania najprawdopodobniej zostaną wykonane podczas pierwszego cyklu ładowania, pozostawiając procesor bez zajęcia do zakończenia ładowania.

Zależności proceduralne

Jak stwierdziliśmy w rozdz. 12, obecność rozgałęzień w sekwencji rozkazów komplikuje działanie potoku. Rozkazy następujące po rozgałęzieniu (dokonanym lub nie dokonanym) wykazują zależność proceduralną od rozgałęzienia i nie mogą być wykonywane przed zakończeniem rozgałęzienia. Na rysunku 14.3 jest pokazany wpływ rozgałęzienia na potok superskalarny stopnia 2.

Jak widzieliśmy, ten rodzaj zależności proceduralnej oddziałuje także na potok skalarny. Jednak podobnie jak poprzednio, konsekwencje w odniesieniu do potoku superskalarnego są poważniejsze, ze względu na większe możliwości tracone przy każdym opóźnieniu.

Jeśli są używane rozkazy o zmiennej długości, to powstaje inny rodzaj zależności proceduralnej. Ponieważ długość określonego rozkazu nie jest znana, musi on być przynajmniej częściowo zdekodowany, zanim będzie mógł być pobrany następny rozkaz. Nie jest potrzebne jednoczesne pobieranie wymagane przez potok superskalarny. Jest to jeden z powodów, dla których metody superskalarne są łatwiejsze do zastosowania w architekturach RISC lub zbliżonych do RISC, które mają ustalone długości rozkazów.

Konflikt dotyczący zasobów

Konflikt dotyczący zasobów polega na jednoczesnym rywalizowaniu dwóch lub wielu rozkazów o te same zasoby. Przykładami takich zasobów są pamięci, pamięci podręczne, magistrale, porty tablic rejestrów i jednostki funkcjonalne (np. sumator ALU).

W odniesieniu do potoku konflikt dotyczący zasobów wygląda podobnie jak w przypadku zależności danych (rys. 14.3). Istnieją jednak pewne różnice. Po pierwsze, problemy dotyczące zasobów mogą być przezwyciężone przez powielenie zasobów, podczas gdy prawdziwa zależność danych nie może być wyeliminowana. Po drugie, gdy czas trwania operacji jest długi, konflikt dotyczący zasobów może być zminimalizowany przez zastosowanie przetwarzania potokowego w odpowiedniej jednostce funkcjonalnej.

14.2. Problemy projektowania

Paralelizm na poziomie rozkazu i paralelizm na poziomie maszyny

W pracy [JOUP89a] istnieje rozróżnienie między dwiema powiązanymi koncepcjami paralelizmu na poziomie rozkazu i paralelizmu na poziomie maszyny. **Parale-**

lizm na poziomie rozkazu występuje, gdy rozkazy w sekwencji są niezależne i wobec tego mogą być wykonywane równolegle przez nakładanie (*overlapping*).

Jako przykład koncepcji paralelizmu na poziomie rozkazu rozważmy dwa następujące fragmenty kodów [JOUP89b]:

```
Load  R1 ← R2          Add    R3 ← R3, „1"
Add   R3 ← R3, „1"     Add    R4 ← R3, R2
Add   R4 ← R4, R2      Store  [R4] ← R0
```

Trzy rozkazy po lewej stronie są niezależne i teoretycznie wszystkie trzy mogłyby być wykonywane równolegle. W przeciwieństwie do tego trzy rozkazy po prawej stronie nie mogą być wykonywane równolegle, ponieważ drugi rozkaz używa wyników pierwszego, a trzeci używa wyników drugiego.

Paralelizm na poziomie rozkazu jest określony przez częstość prawdziwych zależności danych oraz zależności proceduralnych w kodzie. Z kolei te czynniki są zależne od architektury listy rozkazów i od zastosowania. Paralelizm na poziomie rozkazu jest również zdeterminowany przez efekt określony w pracy [JOUP89a] jako *opóźnienie operacji* (*operation latency*): czas wymagany do tego, aby wynik rozkazu stał się dostępny jako argument następnego rozkazu. Parametr ten określa, jakie opóźnienie spowoduje zależność danych lub zależność proceduralna.

Paralelizm na poziomie maszyny jest miarą zdolności procesora do wykorzystania paralelizmu na poziomie rozkazu. Paralelizm maszynowy jest zdeterminowany liczbą rozkazów, które mogą być pobrane i wykonane w tym samym czasie (liczba równoległych potoków), oraz szybkością i złożonością mechanizmów, których procesor używa do znajdywania niezależnych rozkazów.

Oba rodzaje paralelizmu są ważnymi czynnikami zwiększania wydajności. Program może nie wykazywać dostatecznego paralelizmu na poziomie rozkazu, aby w pełni była wykorzystana zaleta paralelizmu maszynowego. Użycie architektury, w której lista rozkazów ma ustaloną długość, takiej jak RISC, umożliwia zwiększenie paralelizmu na poziomie rozkazu. Z drugiej strony ograniczony paralelizm maszynowy ograniczy wydajność niezależnie od natury programu.

Strategia wydawania rozkazów

Jak już stwierdziliśmy, paralelizm maszynowy nie jest po prostu sprawą dysponowania wieloma instancjami każdego etapu potoku. Procesor musi również móc identyfikować paralelizm na poziomie rozkazu i koordynować równoległe pobieranie, dekodowanie i wykonywanie rozkazów. W pracy [JOHN91] użyto terminu **wydawanie rozkazów** do określenia inicjowania wykonywania rozkazów w jednostkach funkcjonalnych procesora oraz terminu **polityka wydawania rozkazów** odnoszącego się do protokołu używanego przy wydawaniu rozkazów.

W istocie, procesor próbuje wyprzedzić bieżący punkt wykonywania w celu zlokalizowania rozkazów, które mogą być wprowadzone do potoku i wykonane. Pod tym względem istotne są:

- kolejność pobierania rozkazów;
- kolejność wykonywania rozkazów;
- kolejność, w której rozkazy zmieniają lokacje w rejestrach i w pamięci.

Im bardziej wyrafinowany jest procesor, tym mniej jest on uzależniony od ścisłych powiązań między tymi kolejnościami. Aby osiągnąć maksymalne wykorzystanie różnych elementów potoku, procesor musi zmieniać jedną lub więcej z wymienionych kolejności w stosunku do oryginalnej kolejności występującej przy wykonywaniu ściśle sekwencyjnym. Jedynym ograniczeniem dla procesora jest to, że wynik musi być poprawny. Wobec tego procesor musi się dostosować do różnych zależności i konfliktów, które omówiliśmy powyżej.

Ogólnie rzecz biorąc, superskalarne strategie wydawania rozkazów możemy podzielić na następujące kategorie:

- kolejne wydawanie połączone z kolejnym kończeniem;
- kolejne wydawanie połączone z kończeniem w zmienionej kolejności;
- wydawanie w zmienionej kolejności połączone z kończeniem w zmienionej kolejności.

Kolejne wydawanie połączone z kolejnym kończeniem

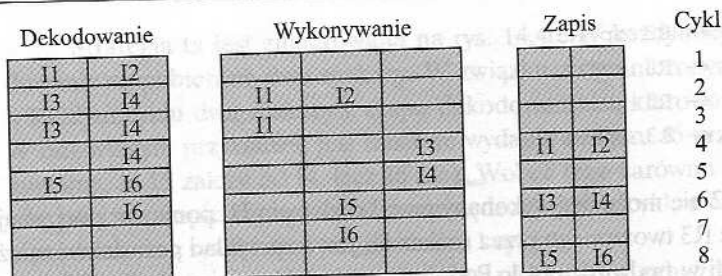
Najprostszą strategią wydawania rozkazów jest wydawanie ich w dokładnej kolejności zgodnej z wykonywaniem sekwencyjnym (kolejne wydawanie) i zapisywanie wyników w tej samej kolejności (kolejne kończenie). Nawet skalarne potoki nie dopuszczają tak uproszczonej strategii. Jest jednak użyteczne rozważenie tej strategii jako podstawy do porównywania bardziej wyrafinowanych rozwiązań.

Na rysunku 14.4a widać przykład takiej strategii. Załóżmy, że potok superskalarny może jednocześnie pobierać i zdekodować dwa rozkazy, że ma trzy odrębne jednostki funkcjonalne (np. jednostkę arytmetyki liczb całkowitych i arytmetyki zmiennopozycyjnej) i że ma dwa przypadki etapu zapisywania. Załóżmy też następujące ograniczenia dotyczące 6-rozkazowego fragmentu programu:

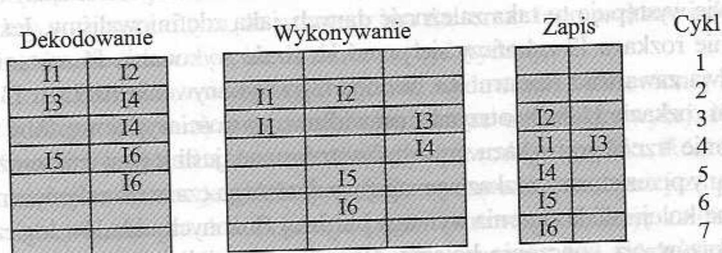
- ❑ I1 wymaga do wykonania 2 cykli.
- ❑ I3 i I4 rywalizują o tę samą jednostkę funkcjonalną.
- ❑ I5 zależy od wartości tworzonej przez I4.
- ❑ I5 i I6 rywalizują o jednostkę funkcjonalną.

Rozkazy są pobierane jednocześnie po dwa i przechodzą do jednostki dekodowania. Ponieważ rozkazy są pobierane parami, następne dwa rozkazy muszą czekać, aż para etapów dekodowania w potoku będzie pusta. Aby zapewnić jednoczesne kończenie, wydawanie rozkazów ulega zatrzymaniu, gdy występuje konflikt dotyczący jednostki funkcjonalnej lub gdy jednostka funkcjonalna wymaga więcej niż jednego cyklu do generowania wyniku.

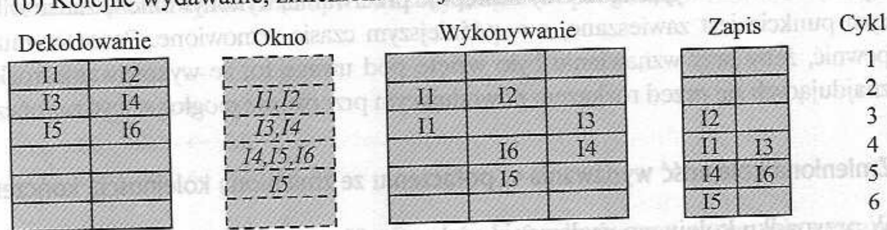
W tym przykładzie czas między dekodowaniem pierwszego rozkazu a zapisaniem ostatniego wyniku wynosi 8 cykli.



(a) Kolejne wydawanie i kolejne kończenie



(b) Kolejne wydawanie, zmieniona kolejność kończenia



(c) Zmieniona kolejność wydawania, zmieniona kolejność kończenia

Rysunek 14.4. Superskalarne metody wydawania i kończenia rozkazów

Kolejne wydawanie połączone ze zmienioną kolejnością kończenia

W skalarnych procesorach RISC zmieniona kolejność kończenia jest stosowana w celu poprawienia wydajności rozkazów wymagających wielu cykli. Na rysunku 14.4b jest pokazane zastosowanie tej strategii w procesorze superskalarnym. Rozkaz I2 może być zakończony przed I1. Umożliwia to wcześniejsze zakończenie rozkazu I3, co netto daje korzyść w postaci zaoszczędzenia jednego cyklu.

Przy zmienionej kolejności kończenia w określonym czasie na etapie wykonywania może się znajdować dowolna liczba rozkazów, aż do osiągnięcia maksymalnego stopnia paralelizmu maszynowego we wszystkich jednostkach funkcjonalnych. Wydawanie rozkazów jest zatrzymywane w wyniku konfliktu dotyczącego zasobów, zależności danych lub zależności proceduralnej.

Poza powyższymi ograniczeniami powstaje nowy rodzaj zależności, który określaliśmy wcześniej jako **zależność wyjściowa** (nazywana także **zależnością odczyt-zapis**). Następujący fragment kodu ilustruje tę zależność (op oznacza dowolną operację):

```
I1:  R3 ← R3 op R5
I2:  R4 ← R3 + 1
I3:  R3 ← R5 + 1
I4:  R7 ← R3 op R4
```

Rozkaz I2 nie może być wykonany przed rozkazem I1, ponieważ potrzebuje wyniku w rejestrze R3 tworzonego przez rozkaz I1; jest to przykład prawdziwej zależności danych, opisanej w podrozdz. 14.1. Podobnie rozkaz I4 musi czekać na I3, ponieważ używa wyniku tworzonego przez I3. Co można powiedzieć o zależności między rozkazami I1 a I3? Nie występuje tu taka zależność danych, jaką zdefiniowaliśmy. Jeśli jednak wykonywanie rozkazu I3 zakończy się przed I1, to do wykonania I4 zostanie pobrana niewłaściwa zawartość rejestru R3. Wobec tego wykonywanie rozkazu I3 musi zakończyć się po rozkazie I1, żeby otrzymać prawidłowe wartości wyjściowe. Aby to zapewnić, wydawanie trzeciego rozkazu musi być wstrzymane, jeśli jego wynik może być później skasowany przez starszy rozkaz wymagający dłuższego czasu do zakończenia.

Zmieniona kolejność kończenia wymaga bardziej złożonych układów logicznych wydawania rozkazów niż kończenie kolejne. Ponadto trudniejsze jest postępowanie z przerwaniami i wyjątkami. Gdy następuje przerwanie, wykonywanie rozkazu w bieżącym punkcie jest zawieszane, a w późniejszym czasie wznowione. Procesor musi zapewnić, żeby przy wznawianiu było wzięte pod uwagę to, że wykonywanie rozkazów znajdujących się przed rozkazem powodującym przerwanie mogło już być zakończone.

Zmieniona kolejność wydawania w połączeniu ze zmienioną kolejnością kończenia

W przypadku kolejnego wydawania rozkazów procesor jedynie dekoduje rozkazy aż do wystąpienia zależności lub konfliktu. Żadne nowe rozkazy nie są dekodowane, dopóki konflikt nie jest rozstrzygnięty. W rezultacie procesor nie może wyprzedzić punktu konfliktu w celu znalezienia następnych rozkazów, które mogą być niezależne od rozkazów znajdujących się już w potoku i które mogą być z pożytkiem wprowadzone do potoku.

Aby umożliwić wydawanie rozkazów w zmienionej kolejności, konieczne jest rozłączenie etapów dekodowania i wykonywania w potoku. Czyni się to za pomocą bufora nazywanego **oknem rozkazu**. Przy takiej organizacji rozkaz po zdekodowaniu przez procesor jest umieszczany w oknie rozkazu. Dopóki bufor ten nie jest zapełniony, procesor może kontynuować pobieranie i dekodowanie nowych rozkazów. Gdy jednostka funkcjonalna na etapie wykonywania staje się osiągalna, rozkaz może być wydany z okna rozkazu do etapu wykonywania. Może być wydany dowolny rozkaz pod warunkiem, że (1) potrzebuje on określonej jednostki funkcjonalnej, która jest osiągalna, oraz (2) żaden konflikt lub zależność nie blokuje tego rozkazu.

W wyniku zastosowania takiej organizacji procesor może „spoglądać w przyszłość", co pozwala mu na zidentyfikowanie niezależnych rozkazów, które mogą być wprowadzone do etapu wykonywania. Rozkazy są wydawane przez okno rozkazu, przy czym nie ma większego znaczenia oryginalny porządek programu. Podobnie jak poprzednio, jedynym ograniczeniem jest prawidłowe wykonanie programu.

Strategia ta jest zilustrowana na rys. 14.4c. W każdym cyklu do etapu dekodowania są pobierane dwa rozkazy. W związku z ograniczonym rozmiarem bufora w każdym cyklu dwa rozkazy z etapu dekodowania są kierowane do okna rozkazu. W omawianym przykładzie jest możliwe wydanie rozkazu I6 przed rozkazem I5 (pamiętamy, że I5 zależy od I4, lecz I6 nie). Wobec tego zarówno na etapie wykonywania, jak i na etapie zapisu został zaoszczędzony jeden cykl, a całkowita oszczędność w porównaniu z rys. 14.4b wynosi również 1 cykl.

Okno rozkazu zostało pokazane na rys. 14.4c w celu zilustrowania jego roli. Okno to nie jest jednak dodatkowym etapem potoku. Pobyt rozkazu w oknie świadczy po prostu o tym, że procesor ma na temat tego rozkazu informację wystarczającą do zdecydowania, kiedy może on być wydany.

Polityka zmienionej kolejności wydawania i zmienionej kolejności kończenia podlega tym samym ograniczeniom, które opisaliśmy wyżej. Rozkaz nie może być wydany, jeśli narusza to zależność lub powoduje konflikt. Różnica polega na tym, że więcej rozkazów jest dostępnych do wydawania, co zmniejsza prawdopodobieństwo zatrzymania potoku. Powstaje ponadto nowego rodzaju zależność, którą określiliśmy wcześniej jako **antyzależność** (nazywana także **zależnością odczyt-zapis**). Zależność tę ilustruje wcześniej rozważany fragment programu:

```
I1:  R3 ← R3 op R5
I2:  R4 ← R3 + 1
I3:  R3 ← R5 + 1
I4:  R7 ← R3 op R4
```

Wykonywanie rozkazu I3 nie może być zakończone, zanim nie zostanie rozpoczęte wykonywanie rozkazu I2 i nie zostaną pobrane jego argumenty. Dzieje się tak, ponieważ rozkaz I3 aktualizuje rejestr R3, który jest źródłem argumentu dla rozkazu I2. Stosowany jest termin *antyzależność*, ponieważ ograniczenie jest podobne do prawdziwej zależności danych, jednak jest odwrócone: to nie pierwszy rozkaz tworzy wartość używaną przez drugi rozkaz, lecz drugi rozkaz niszczy wartość używaną przez pierwszy rozkaz.

Przemianowanie rejestrów

Stwierdziliśmy, że jeśli dozwolona jest zmiana kolejności wydawania rozkazów i (lub) zmiana kolejności ich kończenia, to stwarza to możliwość wystąpienia zależności wyjściowej i antyzależności. Te rodzaje zależności różnią się od prawdziwej zależności danych i od konfliktów dotyczących zasobów, które odzwierciedlają przepływ danych w programie i sekwencję wykonywania rozkazów. Jednak zależności wyjściowe i antyzależności wynikają z tego, że wartości w rejestrach nie odzwierciedlają już sekwencji wartości dyktowanej przez przebieg programu.

Gdy rozkazy są wydawane i kończone w pierwotnej sekwencji, możliwe jest określenie zawartości każdego rejestru w każdym punkcie wykonywania programu. Gdy natomiast są stosowane metody zmienionej kolejności, na podstawie rozważa-

nia sekwencji rozkazów dyktowanej przez program nie można już w pełni określić zawartości rejestrów w każdej chwili. W wyniku tego poszczególne wartości rywalizują o użycie rejestrów, a procesor musi rozwiązywać te konflikty przez okolicznościowe zatrzymywanie etapu potoku.

Zarówno antyzależności, jak i zależności wyjściowe są przykładami konfliktów dotyczących przechowywania. Wiele rozkazów rywalizuje o użycie tych samych lokacji rejestrowych, generując ograniczenia potoku i zmniejszając przez to wydajność. Problem staje się jeszcze bardziej dotkliwy, gdy stosowane są metody optymalizacji rejestrów (omówione w rozdz. 13), ponieważ metody kompilatorowe zmierzają do maksymalizacji wykorzystania rejestrów, co zwiększa liczbę konfliktów dotyczących przechowywania.

Jedna z metod eliminowania tego rodzaju konfliktów jest oparta na tradycyjnym rozwiązaniu konfliktów dotyczących zasobów: na powielaniu zasobów. W tym kontekście metoda ta jest określana jako **przemianowanie rejestrów**. W istocie rejestry są przypisywane dynamicznie przez procesor i są związane z wartościami potrzebnymi do wykonywania rozkazów w różnych punktach czasowych. Gdy jest tworzona nowa wartość rejestru (np. gdy jest wykonywany rozkaz, którego wynik ma być docelowo umieszczony w rejestrze), wartości tej jest przypisywany nowy rejestr. Następne rozkazy, które sięgają do tej wartości jako do argumentu źródłowego w tym rejestrze, muszą przejść przez proces przemianowania: odniesienia do rejestrów w tych rozkazach muszą być zrewidowane, aby dotyczyły rejestrów zawierających potrzebne wartości. Wobec tego, takie same oryginalne odniesienia do rejestru w kilku różnych rozkazach mogą dotyczyć różnych rejestrów rzeczywistych, jeśli chodzi o różne wartości.

Rozważmy, jak można użyć przemianowania rejestrów w stosunku do analizowanego przez nas fragmentu programu:

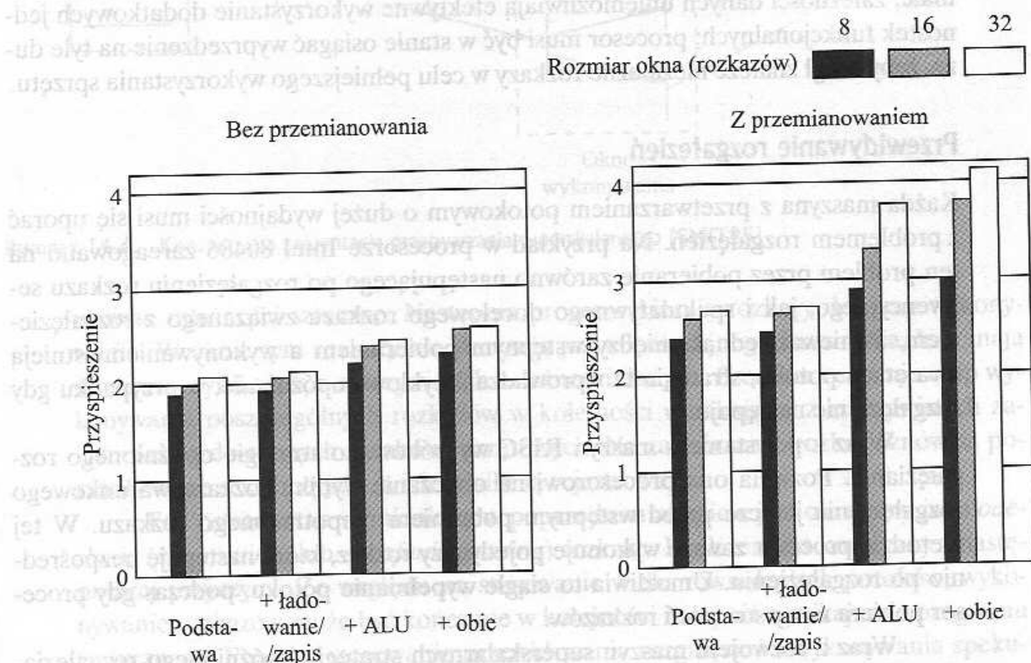
I1: $R3_b \leftarrow R3_a$ op $R5_a$
I2: $R4_b \leftarrow R3_b + 1$
I3: $R3_c \leftarrow R5_a + 1$
I4: $R7_b \leftarrow R3_c$ op $R4_b$

Odniesienie do rejestru bez indeksu oznacza odniesienie logiczne pochodzące z rozkazu. Odniesienie do rejestru z indeksem określa rejestr rzeczywisty przypisany do przechowywania nowej wartości. Gdy jest dokonywany nowy przydział w stosunku do określonego rejestru logicznego, odniesienie do tego samego rejestru logicznego pochodzące z następnego rozkazu przypisuje się najpóźniej przydzielonemu rejestrowi rzeczywistemu (najpóźniej w sensie sekwencji rozkazów programu).

W omawianym przykładzie utworzenie rejestru $R3_c$ w ramach rozkazu I3 zapobiega antyzależności w drugim rozkazie oraz zależności wyjściowej w pierwszym. Nie koliduje ona z prawidłową wartością wymaganą przez I4. W rezultacie rozkaz I3 może być wydany natychmiast; bez przemianowania I3 nie mógłby być wydany przed zakończeniem pierwszego rozkazu i wydaniem drugiego.

Paralelizm maszynowy

Przedyskutowaliśmy dotychczas trzy metody sprzętowe, które mogą być użyte w procesorze superskalarnym w celu zwiększenia wydajności: dublowanie zasobów, zmiana kolejności wydawania rozkazów i przemianowanie. Badanie zależności między tymi metodami przedstawiono w [SMIT89]. W badaniach tych posłużono się symulacją maszyny o własnościach MIPS R2000, uzupełnionej o różne własności superskalarne. Symulowano wiele różnych sekwencji programowych.



Rysunek 14.5. Przyspieszenie wynikające z różnic w organizacji komputerów, bez zależności proceduralnych

Wyniki są pokazane na rys. 14.5. Na każdym wykresie oś pionowa odpowiada przeciętnemu przyspieszeniu maszyny superskalarnej w stosunku do skalarnej. Na osi poziomej wymieniono cztery alternatywne organizacje procesora. W maszynie podstawowej nie zdublowano żadnej jednostki funkcjonalnej, możliwa jest jednak zmiana porządku wydawania rozkazów. Druga konfiguracja zawiera zdublowaną jednostkę ładowania zapisu, która realizuje dostęp do pamięci podręcznej danych. W trzeciej konfiguracji zdublowano ALU, a w czwartej zarówno jednostkę ładowania/zapisu, jak i ALU. Na każdym wykresie pokazano wyniki odnoszące się do okna mieszczącego 8, 16 i 32 rozkazy, z czego wynika wyprzedzenie możliwe do uzyskania przez procesor. Różnica między obydwoma wykresami polega na tym, że w drugim dozwolone jest przemianowanie rejestrów. Oznacza to, że pierwszy wykres dotyczy maszyny ograniczanej przez wszelkie zależności, podczas gdy drugi odpowiada maszynie ograniczanej tylko przez prawdziwe zależności.

Oba wykresy potraktowane łącznie prowadzą do pewnych ważnych konkluzji. Pierwszą z nich jest to, że prawdopodobnie nie warto dodawać jednostki funkcjonalnej bez przemianowania rejestrów. Istnieje wprawdzie pewna niewielka poprawa wydajności, jednak jest ona obciążona wzrostem złożoności sprzętu. W połączeniu z przemianowaniem rejestrów, które eliminuje antyzależności i zależności wyjściowe, dodawanie jednostek funkcjonalnych pozwala na osiąganie istotnej poprawy. Zauważmy jednak, że istnieje znaczna różnica w stopniu poprawy między oknem rozkazu równym 8 a większymi oknami. Wskazuje to, że jeśli okno rozkazu jest zbyt małe, zależności danych uniemożliwiają efektywne wykorzystanie dodatkowych jednostek funkcjonalnych; procesor musi być w stanie osiągać wyprzedzenie na tyle duże, żeby mógł znaleźć niezależne rozkazy w celu pełniejszego wykorzystania sprzętu.

Przewidywanie rozgałęzień

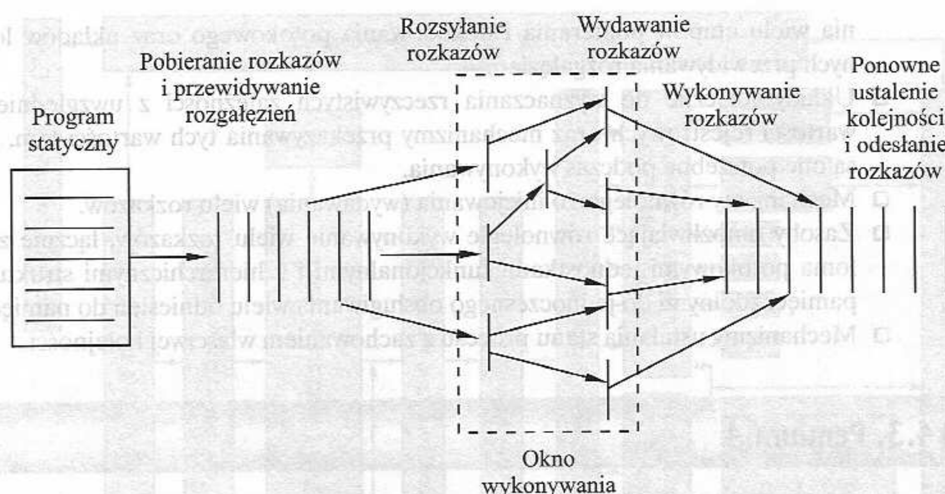
Każda maszyna z przetwarzaniem potokowym o dużej wydajności musi się uporać z problemem rozgałęzień. Na przykład w procesorze Intel 80486 zareagowano na ten problem przez pobieranie zarówno następującego po rozgałęzieniu rozkazu sekwencyjnego, jak i spekulatywnego docelowego rozkazu związanego z rozgałęzieniem. Ponieważ jednak między wstępnym pobieraniem a wykonywaniem istnieją dwa etapy potoku, strategia ta wprowadza 2-cyklowe opóźnienie, w przypadku gdy rozgałęzienie następuje.

Wraz z powstaniem maszyn RISC wypróbowano strategię opóźnionego rozgałęziania. Pozwala ona procesorowi na obliczanie wyniku rozkazu warunkowego rozgałęzienia jeszcze przed wstępnym pobraniem niepotrzebnego rozkazu. W tej metodzie procesor zawsze wykonuje pojedynczy rozkaz, który następuje bezpośrednio po rozgałęzieniu. Umożliwia to ciągłe wypełnianie potoku, podczas gdy procesor pobiera nowy strumień rozkazów.

Wraz z rozwojem maszyn superskalarnych strategia opóźnionego rozgałęziania straciła część uroku. Stało się tak dlatego, że wiele rozkazów wymaga wykonywania w przedziale czasowym związanym z opóźnieniem, co stwarza kilka problemów wynikających z zależności rozkazów. Wobec tego maszyny superskalarne wróciły do poprzedzających RISC metod przewidywania rozgałęzień. Niektóre z nich, jak PowerPC 601, stosują prostą metodę statycznego przewidywania rozgałęzień. Bardziej złożone procesory, takie jak PowerPC 620 i Pentium, stosują dynamiczne przewidywanie rozgałęzień oparte na analizie historii rozgałęzień.

Superskalarne wykonywanie programów

Jesteśmy teraz w stanie dokonać przeglądu superskalarnego wykonywania programów; zostało ono zilustrowane na rys. 14.6. Program, który ma być wykonany, składa się z liniowego szeregu rozkazów. Jest to program statyczny taki, jaki jest pisany przez programistę lub generowany przez kompilator. Proces pobierania rozkazów obejmujący przewidywanie rozgałęzień służy do formowania dynamicznego strumienia rozkazów. Strumień ten jest analizowany pod kątem zależności, a procesor może



Rysunek 14.6. Koncepcyjna prezentacja przetwarzania superskalarnego [SMIT95]

usuwać zależności sztuczne. Następnie procesor kieruje rozkazy do okna wykonywania. W oknie tym rozkazy nie stanowią już sekwencyjnego strumienia, lecz mają strukturę wynikającą z rzeczywistych zależności danych. Procesor realizuje etap wykonywania poszczególnych rozkazów w kolejności wynikającej z rzeczywistych zależności i z dostępnych zasobów sprzętowych. Na zakończenie rozkazy znów są porządkowane sekwencyjnie, a ich wyniki są rejestrowane.

Ten ostatni etap wymieniony w poprzednim akapicie jest określany jako *odesłanie* (*committing*) lub *wycofanie* (*retiring*) rozkazu. Krok ten jest konieczny z następującej przyczyny. Ze względu na stosowanie wielu równoległych potoków, wykonywanie rozkazów może być kończone w kolejności innej niż wynikająca z programu statycznego. Ponadto stosowanie przewidywania rozgałęzień i wykonywania spekulatywnego oznacza, że wykonywanie pewnych rozkazów może być ukończone, po czym są one porzucane, ponieważ reprezentowane przez nie rozgałęzienie nie następuje. Dlatego rejestry trwałego przechowywania i rejestry widzialne dla programu nie mogą być aktualizowane natychmiast po wykonaniu rozkazów. Wyniki muszą być trzymane w pewnego rodzaju pamięci tymczasowej, gdzie są dostępne dla rozkazów zależnych, po czym są one zapisywane trwale dopiero wówczas, gdy zostanie stwierdzone, że rozkazy te zostały wykonane w modelu sekwencyjnym.

Implementacja superskalarna

Na podstawie naszej dotychczasowej dyskusji możemy poczynić pewne ogólne uwagi dotyczące sprzętowych rozwiązań procesora, jakie są wymagane przy podejściu superskalarnym. W pracy [SMIT95] wymieniono następujące podstawowe elementy:

- Strategia jednoczesnego pobierania wielu rozkazów, często na podstawie przewidywanego wyniku rozgałęzień warunkowych. Funkcje te wymagają zastosowa-

nia wielu etapów pobierania i dekodowania potokowego oraz układów logicznych przewidywania rozgałęzień.

- Układy logiczne do wyznaczania rzeczywistych zależności z uwzględnieniem wartości rejestrowych oraz mechanizmy przekazywania tych wartości tam, gdzie są one potrzebne podczas wykonywania.
- Mechanizmy równoległego inicjowania (wydawania) wielu rozkazów.
- Zasoby umożliwiające równoległe wykonywanie wielu rozkazów, łącznie z wieloma potokowymi jednostkami funkcjonalnymi i z hierarchicznymi strukturami pamięci zdolnymi do jednoczesnego obsługiwania wielu odniesień do pamięci.
- Mechanizmy ustalania stanu procesu z zachowaniem właściwej kolejności.

14.3. Pentium 4

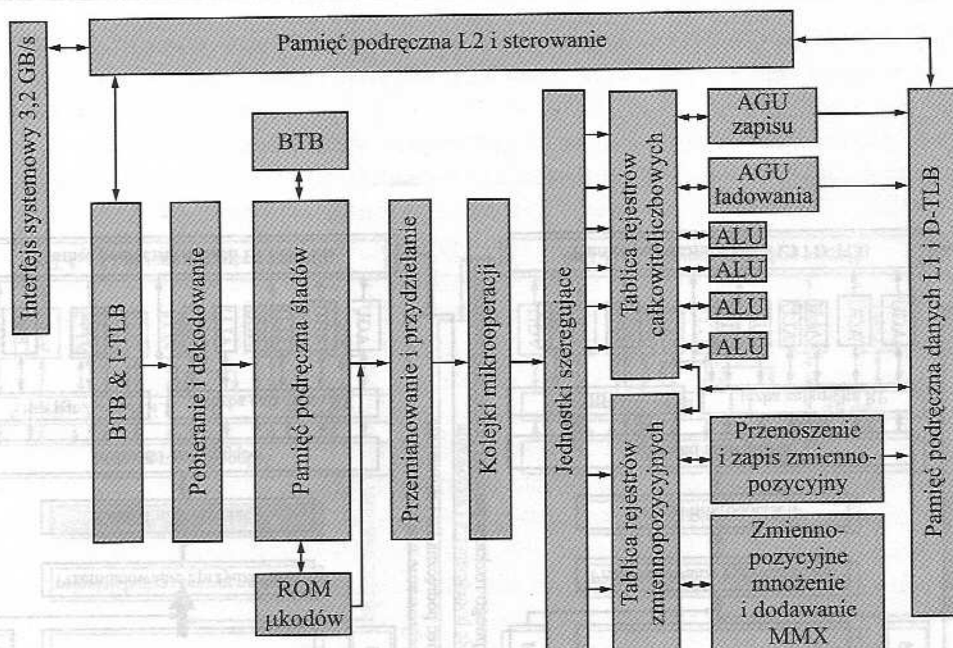
Chociaż koncepcja projektowania superskalarnego jest ogólnie związana z architekturą RISC, takie same zasady superskalarne mogą być zastosowane do maszyn CISC. Być może najbardziej godnym uwagi przykładem takiej możliwości jest Pentium. Ewolucja koncepcji superskalarnych w linii procesorów Intela jest interesująca. 80486 był prostą, tradycyjną maszyną CISC, bez elementów superskalarnych. Pierwszy z procesorów Pentium zawierał skromne elementy superskalarne w postaci dwóch odrębnych jednostek wykonywania operacji całkowitoliczbowych. W Pentium Pro rozwiązania superskalarne zostały wprowadzone na pełną skalę. Następne modele Pentium zawierały udoskonalone i poszerzone rozwiązania superskalarne.

Ogólny schemat blokowy Pentium 4 został pokazany na rys. 4.13. Rysunek 14.7, oparty na [CARM00], ukazuje tę samą strukturę w sposób odpowiedniejszy do analizowania przetwarzania potokowego, którym zajmiemy się w tym podrozdziale. Działanie Pentium 4 można podsumować następująco:

1. Procesor pobiera rozkazy z pamięci w kolejności zgodnej z programem statycznym.
2. Każdy rozkaz jest tłumaczony na jeden lub wiele rozkazów RISC o ustalonej długości, znanych jako mikrooperacje.
3. Procesor wykonuje mikrooperacje, używając superskalarnej organizacji potokowej, dzięki czemu mogą one być wykonywane w zmienionej kolejności.
4. Procesor odsyła wyniki każdej mikrooperacji do własnego zbioru rejestrów, układając je zgodnie z przebiegiem programu początkowego.

Architektura Pentium 4 składa się z zewnętrznej powłoki CISC z wewnętrznym rdzeniem RISC. Wewnętrzne mikrooperacje RISC przechodzą przez potok o co najmniej 20 etapach (rys. 14.8); w niektórych przypadkach mikrooperacja wymaga wielu etapów wykonywania, czego konsekwencją jest jeszcze dłuższy potok. Stanowi to kontrast w stosunku do 5-etapowego potoku (rys. 12.18) używanego w procesorach Intel x86 i w Pentium.

Prześledzimy teraz działanie potoku Pentium 4, posługując się rys. 14.9 jako ilustracją.



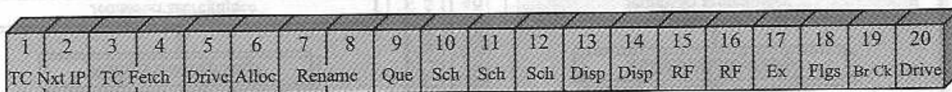
AGU = *address generation unit* – jednostka generowania adresu
BTB = *branch target buffer* – bufor celów rozgałęzień
D-TLB = *data translation lookaside buffer* – bufor translacji adresów tablic stron rozkazów
I-TLB = *instruction translation lookaside buffer* – bufor translacji adresów tablic stron rozkazów

Rysunek 14.7. Schemat blokowy Pentium 4

Część czołowa

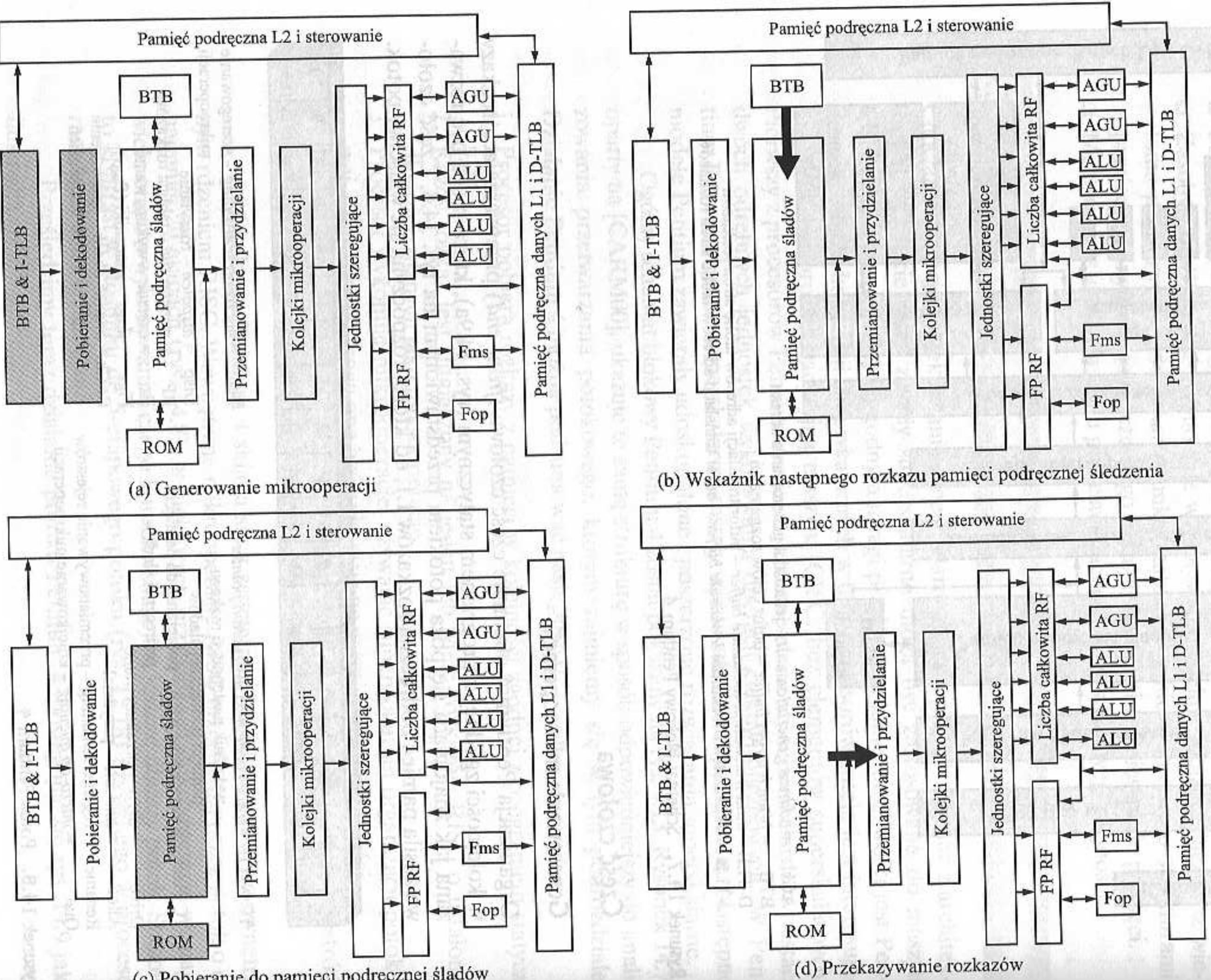
Generowanie mikrooperacji

Organizacja Pentium 4 obejmuje część czołową (*front end*) przetwarzającą rozkazy w kolejności zgodnej z programem statycznym (rys. 14.9a), która może być rozważana jak znajdująca się poza potokiem przedstawionym na rys. 14.8. Część czołowa zasila pamięć podręczną rozkazów L1, od której rozpoczyna się właściwy potok.

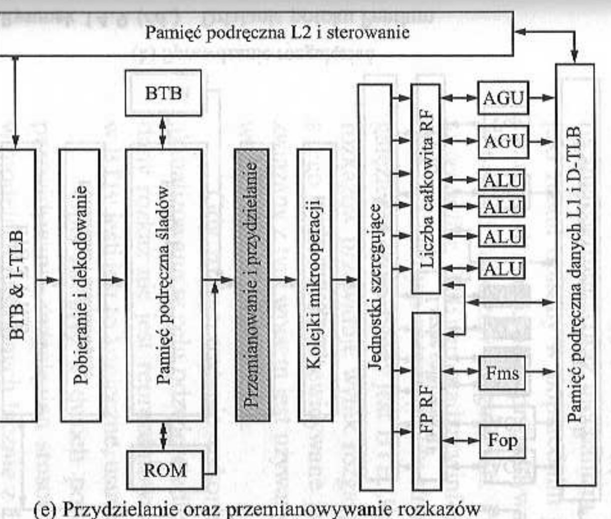


TC Nxt IP = *trace cache next instruction pointer* – wskaźnik następnego rozkazu pamięci podręcznej śladów
TC Fetch = *trace cache fetch* – pobieranie do pamięci podręcznej śladów
Drive = *drive* – przekazywanie
Alloc = *allocate* – przydział
Rename = *register renaming* – przemianowywanie rejestrów
Que = *micro-op queuing* – kolejkowanie mikrooperacji
Sch = *micro-op scheduling* – szeregowanie mikrooperacji
Disp = *dispatch* – rozsyłanie
RF = *register file* – tablica rozkazów
Ex = *execute* – wykonywanie
Flgs = *flags* – znaczniki (flagi)
Br Ck = *branch check* – sprawdzenie rozgałęzień

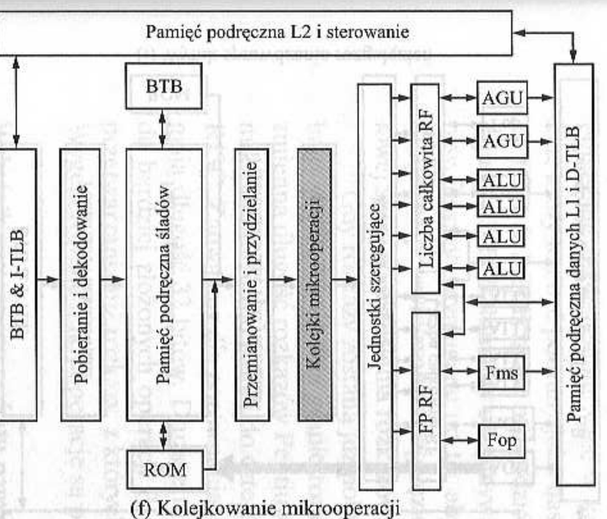
Rysunek 14.8. Potok Pentium 4



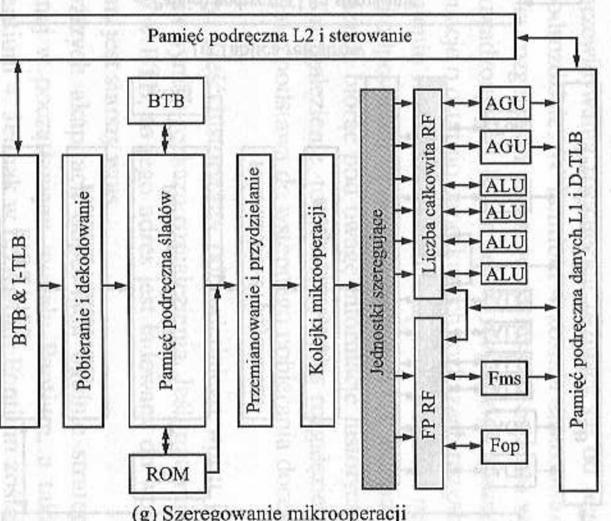
Rysunek 14.9. Działanie potoku Pentium



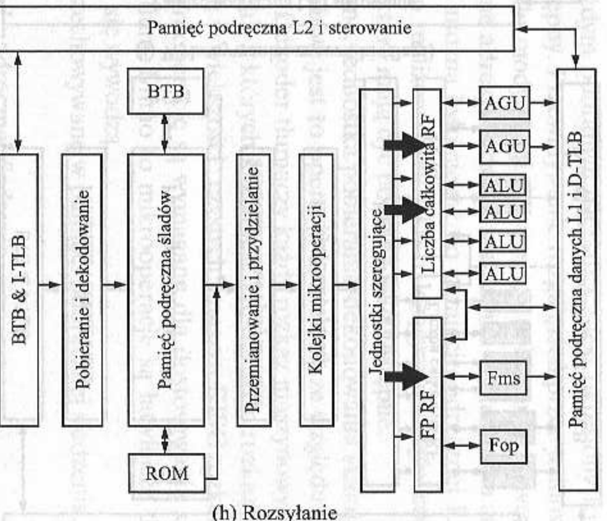
(e) Przydzielanie oraz przemianowywanie rozkazów



(f) Kolejkowanie mikrooperacji

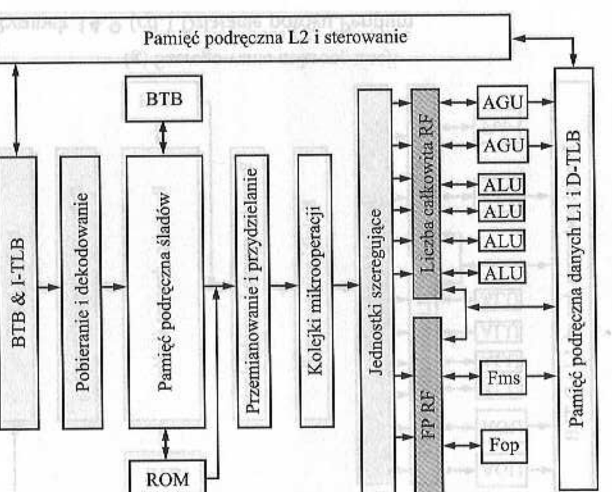


(g) Szeregowanie mikrooperacji

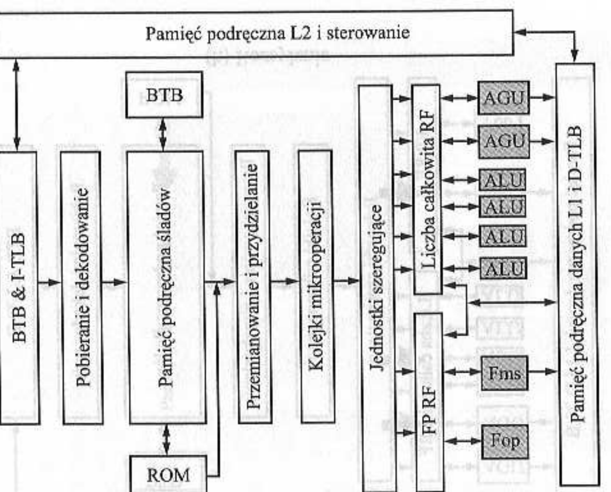


(h) Rozsyłanie

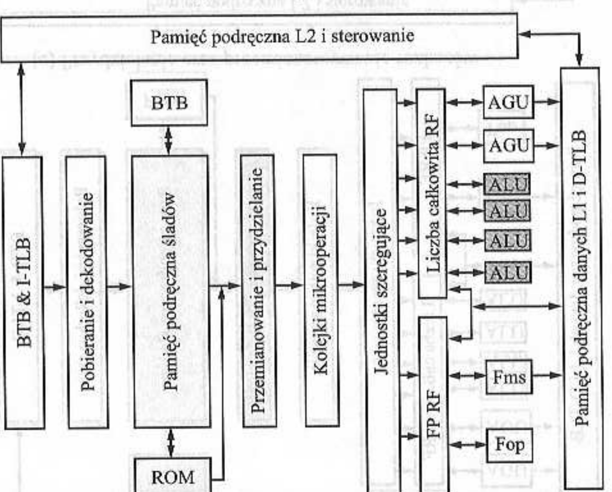
Rysunek 14.9 (cd.) Działanie potoku Pentium



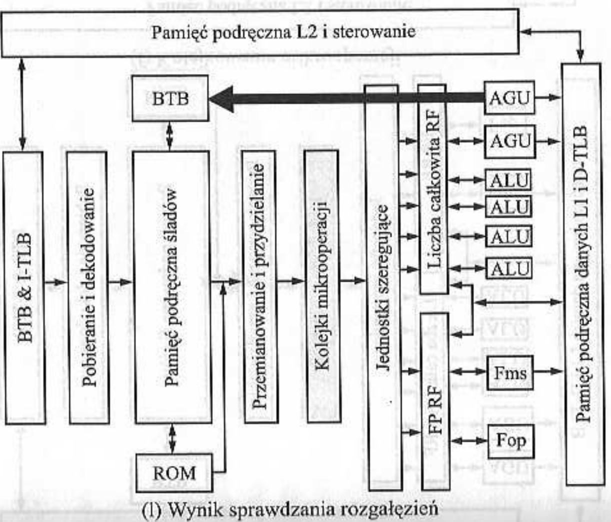
(i) Tablica rejestrów



(j) Wykonywanie; znaczniki (flagi)



(k) Sprawdzanie rozgałęzień



(l) Wynik sprawdzania rozgałęzień

Rysunek 14.9 (cd.) Działanie potoku Pentium

Zwykle procesor działa na podstawie zawartości pamięci podręcznej śladów (*trace cache*); gdy w tej pamięci następuje chybienie, część czołowa wprowadza do niej nowe rozkazy.

Za pomocą bufora celów rozgałęzień i bufora translacji adresów tablic stron rozkazów (BTB & I-TLB) jednostka pobierania i dekodowania pobiera rozkazy maszynowe Pentium 4 z pamięci podręcznej L2 po 64 bajty. Domyślnie rozkazy są pobierane sekwencyjnie, więc w każdym pobranym wierszu L2 jest określony rozkaz, który ma być pobrany jako następny. Przewidywanie rozgałęzień przez jednostkę BTB & I-TLB może prowadzić do modyfikacji tej operacji pobierania sekwencyjnego. I-TLB tłumaczy otrzymywane adresy liniowego wskaźnika rozkazów na adresy fizyczne, wymagane po to, aby można było sięgać do pamięci podręcznej L2. Statyczne przewidywanie rozgałęzień realizowane w BTB należącym do części czołowej służy do określania rozkazów, które mają być pobrane jako następne.

Gdy rozkazy zostaną już pobrane, jednostka pobierania-dekodowania skanuje bajty w celu określenia granic rozkazów; jest to operacja konieczna ze względu na zmienną długość rozkazów Pentium. Dekoder tłumaczy każdy rozkaz maszynowy na mikrooperacje (od jednej do czterech), z których każda jest 118-bitowym rozkazem RISC. Zauważmy dla porównania, że w większości „czystych" maszyn RISC rozkazy mają długość 32 bitów. Dłuższe mikrooperacje są wymagane dla dostosowania się do bardziej złożonych operacji Pentium. Mimo to mikrooperacje są łatwiejsze do przetwarzania niż rozkazy, z których się wywodzą.
Wygenerowane mikrooperacje są przechowywane w pamięci podręcznej śledzenia.

Wskaźnik następnego rozkazu pamięci podręcznej śledzenia

Dwa pierwsze etapy potoku (rys. 14.9b) obejmują wybór rozkazów w pamięci podręcznej śledzenia oraz mechanizm przewidywania rozgałęzień różniący się od tego, jaki został opisany w poprzednim podrozdziale. W Pentium 4 została zastosowana strategia dynamicznego przewidywania rozgałęzień opartego na historii ostatnio wykonanych rozkazów rozgałęzień. Prowadzony jest bufor celów rozgałęzień (*branch target buffer*, BTB) gromadzący informacje o ostatnio napotkanych rozkazach rozgałęzień. Za każdym razem, gdy w strumieniu rozkazów zostanie napotkany rozkaz rozgałęzienia, sprawdzany jest BTB. Jeśli odpowiedni wpis istnieje już w BTB, jednostka rozkazów przewiduje wynik rozgałęzienia, biorąc pod uwagę informacje historyczne z tego wpisu. Jeśli przewidywane jest rozgałęzienie, to docelowy adres rozgałęzienia związany z tym wpisem jest używany jako podstawa do wstępnego pobierania docelowego rozkazu rozgałęzienia.

Gdy rozkaz zostanie wykonany, część historyczna odpowiedniego wpisu jest aktualizowana w celu odzwierciedlenia wyniku rozkazu rozgałęzienia. Jeśli natomiast dany rozkaz nie jest reprezentowany w BTB, to jego adres jest ładowany do wpisu w BTB; jeśli jest to konieczne, usuwany jest starszy wpis.

Opis zawarty w dwóch powyższych akapitach odpowiada ogólnie strategii przewidywania rozgałęzień stosowanej w początkowym modelu Pentium, a także w modelach następnych łącznie z Pentium 4. Jednak w przypadku Pentium zostało

zastosowane proste rozwiązanie z 2 bitami określającymi historię. Późniejsze modele Pentium mają znacznie dłuższe potoki (20 etapów w Pentium 4 w porównaniu z 5 etapami w Pentium), więc negatywne konsekwencje błędnego przewidywania są poważniejsze. Dlatego w późniejszych modelach Pentium zastosowano bardziej rozwinięty system przewidywania rozgałęzień z większą liczbą bitów historycznych w celu zmniejszenia udziału błędów przewidywania.

W Pentium 4 BTB ma postać czterodrożnej sekcyjno-skojarzeniowej pamięci podręcznej mieszczącej 512 wierszy. W każdym wpisie adres rozgałęzienia służy jako znacznik. Wpis obejmuje również docelowy adres rozgałęzienia, jaki został wybrany podczas ostatniego wykonywania tego rozgałęzienia, oraz 4-bitowe pole historii. Stanowi to znaczną różnicę w porównaniu z 2 bitami używanymi w pierwszym modelu Pentium i w większości procesorów superskalarnych. Dzięki 4 bitom, przy przewidywaniu rozgałęzień mechanizm Pentium 4 może brać pod uwagę dłuższą historię. Algorytm taki zwykło się określać jak algorytm Yeha [YEH91]. Jego twórcy wykazali, że zapewnia on znaczne ograniczenie błędnych przepowiedni w porównaniu z algorytmami opartymi na 2 bitach historii [EVER98].

Rozgałęzienia warunkowe, których historia nie została wpisana do BTB, są przewidywane za pomocą algorytmu statycznego zgodnego z następującymi zasadami:

- Dla adresów rozgałęzień, które nie są określone względem wskaźnika rozkazów (IP), przewiduje się wystąpienie rozgałęzienia, jeśli jest ono powrotem, oraz brak rozgałęzienia w pozostałych przypadkach.
- Dla warunkowych rozgałęzień wstecznych określonych względem IP przewiduje się wystąpienie rozgałęzienia. Zasada ta odzwierciedla typowe zachowanie pętli.
- Dla rozgałęzień warunkowych skierowanych ku przodowi i określonych względem IP przewiduje się brak rozgałęzienia.

Pobieranie do pamięci podręcznej śladów

Pamięć podręczna śledzenia (rys. 14.9c) pobiera zdekodowane już mikrooperacje z dekodera rozkazów i składa z nich sekwencje o kolejności wynikającej z programu, zwane *śladami*. Mikrooperacje są pobierane z pamięci podręcznej śladów sekwencyjnie i trafiają do układów logicznych przewidywania rozgałęzień.

Kilka rozkazów wymaga więcej niż czterech mikrooperacji. Są one przenoszone do pamięci stałej (ROM) mikrokodów, zawierającej szeregi mikrooperacji (po pięć lub więcej) związane ze złożonymi rozkazami maszynowymi. Na przykład rozkazy operacji na ciągach znaków mogą być tłumaczone na bardzo duże, powtarzalne sekwencje mikrooperacji (których mogą być setki). Pamięć stała mikrokodów jest więc mikroprogramowaną jednostką sterowania w sensie przedstawionym w części 4. Gdy pamięć stała mikrokodów zakończy szeregowanie mikrooperacji dla bieżącego rozkazu Pentium, zostaje wznowione pobieranie z pamięci podręcznej śladów.

Przekazywanie

W piątym etapie potoku Pentium 4 (rys. 14.9d) następuje przekazanie zdekodowanych rozkazów z pamięci podręcznej śladów do modułu przemianowywania i przydzielania.

Układy logiczne wykonywania rozkazów poza kolejnością

W tej części procesora następuje zmiana kolejności mikrooperacji w celu umożliwienia ich szybkiego wykonania, gdy tylko są dostępne argumenty wejściowe.

Przydzielanie

Etap przydzielania (rys. 14.9e) obejmuje przydzielanie zasobów wymaganych do wykonywania rozkazów. W jego skład wchodzą następujące działania:

- Jeśli wymagane zasoby (np. rejestr) są niedostępne dla jednej z trzech mikrooperacji przybywających do jednostki przydzielania podczas cyklu zegara, jednostka ta opóźnia potok.
- Jednostka przydzielania przypisuje wpis bufora zmiany kolejności (*reorder buffer* – ROB) śledzący stan ukończenia jednej ze 126 mikrooperacji, które w każdej chwili mogłyby być wprowadzone do procesu.
- Jednostka przydzielania przypisuje jeden ze 128 wpisów rejestru całkowitoliczbowego lub zmiennopozycyjnego wartości danych wynikowych mikrooperacji oraz (być może) ładuje lub zapisuje bufor służący do śledzenia jednego spośród 48 ładowań lub 24 zapisów znajdujących się w potoku.
- Jednostka przydzielania wprowadza wpis do jednej z dwóch kolejek mikrooperacji poprzedzających układy szeregujące.

ROB jest buforem cyklicznym mieszczącym do 126 mikrooperacji i zawiera również 128 rejestrów sprzętowych. Każdy wpis tego bufora składa się z następujących pól:

- **Stan.** Wskazuje, czy ta mikrooperacja została wprowadzona do szeregu przewidzianego do wykonania, została przesłana do wykonania czy też jej wykonywanie zostało ukończone i jest gotowa do odesłania.
- **Adres pamięci.** Adres rozkazu Pentium, który wygenerował daną mikrooperację.
- **Mikrooperacja.** Rzeczywista operacja.
- **Rejestr zamienny** (*alias*). Jeśli mikrooperacja odnosi się do jednego z 16 rejestrów przewidzianych w architekturze, wpis ten kieruje to odniesienie do jednego ze 128 rejestrów sprzętowych.

Mikrooperacje są wprowadzane do ROB kolejno, następnie są przesyłane do jednostki rozsyłania i wykonywania bez zachowania tej kolejności. Kryterium przesyłania jest to, czy dla danej mikrooperacji są dostępne odpowiednia jednostka wykonywania i wszystkie niezbędne dane. Mikrooperacje są w końcu odsyłane z ROB

z zachowaniem kolejności. W celu zrealizowania odsyłania w kolejności, po wyznaczeniu mikrooperacji gotowych do odesłania najpierw są odsyłane mikrooperacje najstarsze.

Przemianowywanie rejestrów

Na etapie przemianowywania (rys. 14.9e) następuje odwzorowanie odwołań do 16 rejestrów przewidzianych w architekturze (8 zmiennopozycyjnych oraz EAX, EBX, ECX, EDX, ESI, EDI, EBP i ESP) w zbiorze 128 rejestrów fizycznych. Na tym etapie są usuwane fałszywe odniesienia wynikające ze skończonej liczby rejestrów architekturowych przy zachowaniu prawdziwych zależności danych (odczyty po zapisach).

Kolejkowanie mikrooperacji

Po przydzieleniu zasobów i przemianowaniu rejestrów mikrooperacje są umieszczane w jednej z dwóch kolejek mikrooperacji (rys. 14.9f), gdzie pozostają, aż pojawi się miejsce w układach szeregowania. Jedna z dwóch kolejek obejmuje operacje pamięciowe (ładowania i zapisu), druga zaś mikrooperacje nie zawierające odniesień do pamięci. Każda kolejka funkcjonuje zgodnie z regułą FIFO (pierwsza na wejściu, pierwsza na wyjściu), jednak nie ma ustalonej kolejności między kolejkami. Oznacza to, że mikrooperacja z jednej kolejki może być wczytana bez zachowania kolejności w stosunku do mikrooperacji znajdujących się w drugiej kolejce. Pozwala to na bardziej elastyczne działanie układów szeregujących.

Szeregowanie i rozsyłanie mikrooperacji

Jednostki szeregowania (rys. 14.9g) są odpowiedzialne za pobieranie mikrooperacji z kolejek i rozsyłanie ich do wykonania. Każda z tych jednostek wyszukuje mikrooperacje, których stan wskazuje na to, że dysponują wszystkimi argumentami. Jeśli jednostka wykonywania wymagana dla takiej mikrooperacji jest dostępna, jednostka szeregująca pobiera tę mikrooperację i przesyła ją do odpowiedniej jednostki wykonywania (rys. 14.9h). W jednym cyklu może być rozesłanych do 6 mikrooperacji. Jeśli dla danej jednostki wykonywania jest dostępnych wiele mikrooperacji, jednostka szeregująca przesyła je w kolejności wynikającej z kolejki. Jest to zgodne z regułą FIFO faworyzującą kolejne wykonywanie mikrooperacji, jednak tym razem strumień rozkazów został tak przeorganizowany w wyniku zależności i rozgałęzień, że jego porządek został w znacznym stopniu zmieniony.

Cztery porty łączą jednostki szeregujące z jednostkami wykonującymi. Port 0 służy rozkazom zarówno całkowitoliczbowym, jak i zmiennopozycyjnym, z wyjątkiem prostych operacji całkowitoliczbowych i przetwarzania błędnych przewidywań rozgałęzień, które są kierowane do portu 1. Ponadto do tych samych portów są przydzielane jednostki wykonujące MMX. Pozostałe porty służą operacjom pamięciowym ładowania i zapisu.

Jednostki wykonywania: całkowitoliczbowa i zmiennopozycyjna

Źródłami dla operacji oczekujących w jednostkach wykonujących są całkowitoliczbowe i zmiennopozycyjne tablice rejestrów (rys. 14.9i). Jednostki wykonujące pobierają wartości z tablic rejestrów, a także z pamięci podręcznej danych L1 (rys. 14.9j). Odrębny etap potoku służy do obliczania znaczników (np. zero, ujemna); wchodzą one zwykle w skład danych wejściowych dla rozkazów rozgałęzień.

W kolejnym etapie potoku jest realizowane sprawdzanie rozgałęzień (rys. 14.9k). Polega ono na porównywaniu rzeczywistego wyniku rozgałęzienia z przepowiadanym. Jeśli przewidywanie rozgałęzienia okazało się błędne, to na różnych etapach przetwarzania znajdują się mikrooperacje, które muszą być usunięte z potoku. Następnie – podczas etapu przekazywania (rys. 14.9l) – do jednostki przewidywania rozgałęzień jest przekazywane odpowiednie miejsce przeznaczenia rozgałęzienia, co powoduje ponowne uruchomienie potoku, począwszy od nowego adresu docelowego.

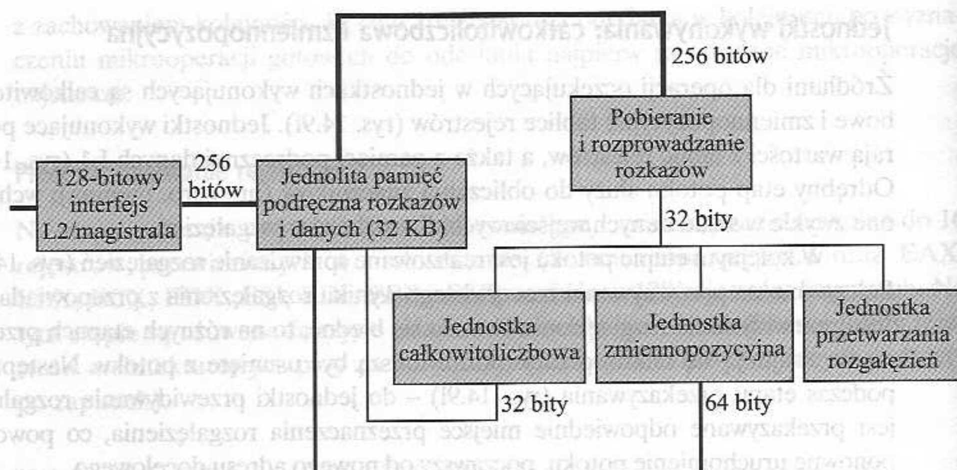
14.4. PowerPC

Architektura PowerPC jest bezpośrednim potomkiem IBM 801, RT PC oraz RS/6000, przy czym ostatnia z tych maszyn była również określana jako wdrożenie architektury POWER. Wszystkie te maszyny były maszynami RISC, jednak pierwszą wykazującą cechy superskalarne była RS/6000. W pierwszym modelu z serii PowerPC, a więc 601, zastosowano rozwiązanie superskalarne bardzo zbliżone do RS/6000. Następne modele PowerPC stanowiły dalsze rozwinięcie koncepcji superskalarnej. W tym podrozdziale skupimy się na modelu 601, który jest dobrym przykładem projektu superskalarnego opartego na RISC. Na koniec krótko przeanalizujemy model 620.

PowerPC 601

Na rysunku 14.10 jest pokazana w zarysie organizacja modelu 601. Podobnie jak inne maszyny superskalarne, procesor 601 składa się z niezależnych jednostek funkcjonalnych w celu zwiększenia możliwości równoległego wykonywania rozkazów. W szczególności rdzeń 601 stanowią trzy niezależne, przetwarzające potokowo jednostki wykonawcze: przetwarzania liczb całkowitych, przetwarzania liczb zmiennopozycyjnych i rozgałęzień. Łącznie jednostki te mogą jednocześnie wykonywać 3 rozkazy, co oznacza rozwiązanie superskalarne stopnia 3.

Na rysunku 14.11 widać schemat logiczny architektury 601; jest tu również pokazany przepływ rozkazów między jednostkami funkcjonalnymi. Jednostka pobierania może jednocześnie wstępnie pobierać z pamięci podręcznej do 8 rozkazów. Jednostka pamięci podręcznej obsługuje połączoną pamięć podręczną zarówno rozkazów, jak i danych. Jest ona odpowiedzialna za doprowadzanie rozkazów do innych jednostek oraz danych do rejestrów. Układy logiczne arbitrażu pamięci podręcznej wysyłają adres o najwyższym priorytecie dostępu do tej pamięci.



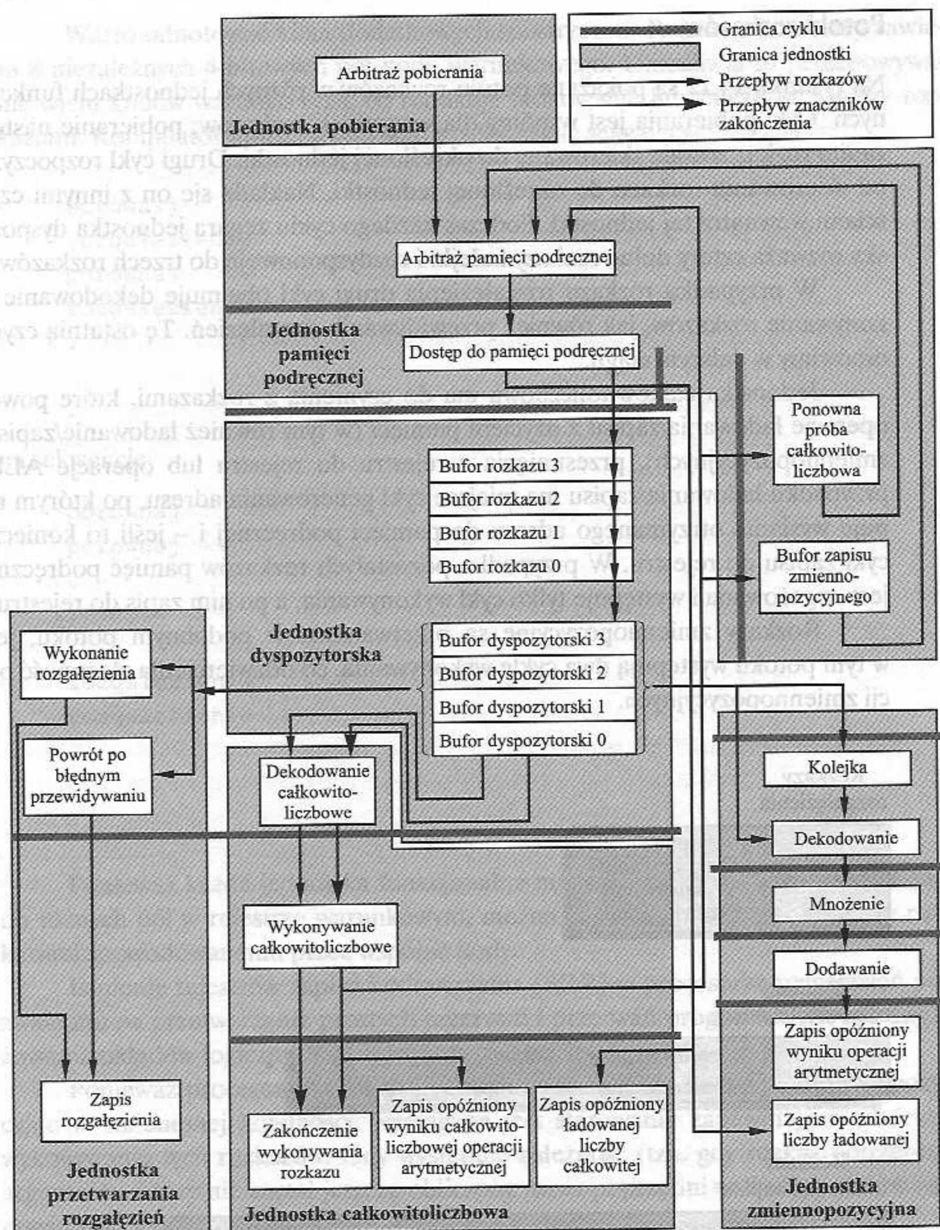
Rysunek 14.10. Schemat blokowy procesora PowerPC 601

Jednostka dyspozytorska

Jednostka dyspozytorska pobiera rozkazy z pamięci podręcznej i ładuje je do kolejki dyspozytorskiej, w której może jednocześnie przebywać 8 rozkazów. Przetwarza ona ten strumień rozkazów, zapewniając ich stały dopływ do jednostek przetwarzania rozgałęzień, liczb całkowitych i zmiennopozycyjnych. Górna połowa kolejki działa po prostu jako bufor przetrzymujący rozkazy, zanim trafią one do dolnej połowy kolejki. Rolą górnej połowy jest zapewnienie tego, żeby jednostka dyspozytorska nie opóźniała się, czekając na rozkazy z pamięci podręcznej. W dolnej połowie kolejki rozkazy są rozprowadzane zgodnie z następującym schematem:

- **Jednostka przetwarzania rozgałęzień.** Przetwarza ona wszystkie rozgałęzienia. Najniżej położony rozkaz rozgałęzienia w dolnej połowie kolejki dyspozytorskiej jest wydawany do przetwarzania, jeśli tylko jednostka ta to akceptuje.
- **Jednostka zmiennopozycyjna.** Wykonuje wszystkie rozkazy zmiennopozycyjne. Rozkaz zmiennopozycyjny najniżej położony w dolnej połowie kolejki dyspozytorskiej jest wydawany do tej jednostki, jeśli jej potok rozkazów nie jest pełny.
- **Jednostka całkowitoliczbowa.** Wykonuje operacje na liczbach całkowitych, operacje ładowania/zapisu między tablicą rejestrów a pamięcią podręczną oraz operacje porównywania liczb całkowitych. Rozkaz dotyczący liczb całkowitych jest wydawany tylko wtedy, kiedy dostanie się on na dno kolejki dyspozytorskiej.

Możliwość wydawania rozkazów rozgałęzienia i rozkazów zmiennopozycyjnych w zmienionym porządku z kolejki dyspozytorskiej pomaga utrzymywać wypełnienie potoków rozkazów w jednostkach przetwarzania rozgałęzień i zmiennopozycyjnej oraz ułatwia szybkie przechodzenie rozkazów przez kolejkę dyspozytorską.



Rysunek 14.11. Struktura potoku procesora PowerPC 601 [POTT94]

Jednostka dyspozytorska zawiera także układy logiczne, które pozwalają jej obliczać adresy wstępnego pobierania. Kontynuuje ona sekwencyjne pobieranie rozkazów, aż rozkaz rozgałęzienia zostanie przeniesiony do dolnej połowy kolejki. Gdy jednostka przetwarzania rozgałęzień przetworzy rozkaz, może ona zaktualizować adres wstępnego pobierania, dzięki czemu następne rozkazy są pobierane spod nowego adresu i wprowadzane do kolejki dyspozytorskiej.

Potoki rozkazów

Na rysunku 14.12 są pokazane potoki rozkazów w różnych jednostkach funkcjonalnych. Cykl pobierania jest wspólny dla wszystkich rozkazów; pobieranie następuje, zanim rozkaz zostaje skierowany do określonej jednostki. Drugi cykl rozpoczyna się od skierowania rozkazu do określonej jednostki. Nakłada się on z innymi czynnościami wewnątrz tej jednostki. Podczas każdego cyklu zegara jednostka dyspozytorska rozważa cztery dolne rozkazy kolejki i rozdysponowuje do trzech rozkazów.

W przypadku rozkazu rozgałęzienia drugi cykl obejmuje dekodowanie i wykonywanie rozkazów, jak również przewidywanie rozgałęzień. Tę ostatnią czynność omówimy w dalszym ciągu.

Jednostka całkowitoliczbowa ma do czynienia z rozkazami, które powodują operacje ładowania/zapisu z użyciem pamięci (w tym również ładowanie/zapis liczb zmiennopozycyjnych), przesunięcia z rejestru do rejestru lub operacje ALU. W przypadku ładowania/zapisu ma miejsce cykl generowania adresu, po którym następuje wysłanie otrzymanego adresu do pamięci podręcznej i – jeśli to konieczne – cykl zapisu do rejestru. W przypadku pozostałych rozkazów pamięć podręczna nie jest angażowana i występuje tylko cykl wykonywania, a po nim zapis do rejestru.

Rozkazy zmiennopozycyjne są przetwarzane w podobnym potoku, jednak w tym potoku występują dwa cykle wykonywania, co odzwierciedla złożoność operacji zmiennopozycyjnych.

Rozkazy rozgałęzienia

Pobieranie	Rozprowadzanie Dekodowanie Wykonywanie Przewidywanie

Rozkazy całkowitoliczbowe

Pobieranie	Rozprowadzenie Dekodowanie	Wykonywanie	Zapis opóźniony

Rozkazy ładowania/zapisu

Pobieranie	Rozprowadzenie Dekodowanie	Generowanie adresu	Pamięć podręczna	Zapis opóźniony

Rozkazy zmiennopozycyjne

Pobieranie	Rozprowadzenie	Dekodowanie	Wykonywanie 1	Wykonywanie 2	Zapis opóźniony

Rysunek 14.12. Potok procesora PowerPC 601

Warto odnotować kilka dodatkowych spostrzeżeń. Rejestr warunkowy zawiera 8 niezależnych 4-bitowych pól kodu warunkowego. Umożliwia to przechowywanie wielu kodów warunkowych, co redukuje blokowanie lub zależność między rozkazami. Kompilator może na przykład przekształcić sekwencję

```
porównaj
rozgałęzienie
porównaj
rozgałęzienie
.
.
.
```

na sekwencję

```
porównaj
porównaj
.
.
.
rozgałęzienie
rozgałęzienie
.
.
.
```

Ponieważ każda jednostka funkcjonalna może wysłać swoje kody warunkowe do różnych pól w rejestrze warunkowym, można zapobiec blokowaniu między rozkazami spowodowanemu przez wspólne kody.

Istnienie rejestrów zapisu i odtworzenia (SRR) w procesorze rozgałęzień pozwala mu na przetwarzanie prostych przerwań i przerwań programowych bez angażowania układów logicznych innych jednostek funkcjonalnych.

Ponieważ procesor 601 może wydawać rozkazy rozgałęzienia i zmiennopozycyjne w zmienionej kolejności, wymagane jest sterowanie zapewniające właściwe wykonywanie tych rozkazów. Gdy występuje zależność (tzn. gdy rozkaz potrzebuje argumentu, który nie został jeszcze obliczony przez poprzedni rozkaz), potok w odpowiedniej jednostce funkcjonalnej ulega zatrzymaniu.

Przetwarzanie rozgałęzień

Kluczem do wysokiej wydajności maszyny RISC lub superskalarnej jest jej zdolność do optymalizacji wykorzystania potoku. Zwykle najbardziej krytycznym elementem projektowania jest sposób przetwarzania rozgałęzień. W maszynie PowerPC za przetwarzanie rozgałęzień jest odpowiedzialna jednostka rozgałęzień. Jednostka ta została zaprojektowana w taki sposób, że w wielu przypadkach rozgałęzienia nie

mają wpływu na przebieg wykonywania rozkazów w innych jednostkach. W celu osiągnięcia zerocyklowego rozgałęziania zastosowano następujące strategie:

1. Wbudowano układy logiczne przeglądające bufor dyspozytorski w poszukiwaniu rozgałęzień. Generowane są docelowe adresy rozgałęzienia, gdy rozgałęzienie pojawia się po raz pierwszy w dolnej części kolejki i gdy żadne poprzednie rozgałęzienia nie zawiesiły wykonywania.
2. Dokonywana jest próba określenia wyniku rozgałęzienia warunkowego. Jeśli kod warunkowy został ustawiony z dostatecznym wyprzedzeniem, to wynik ten może być określony. W każdym przypadku tuż po napotkaniu rozkazu rozgałęzienia układy logiczne określają:
   (a) Czy rozgałęzienie nastąpi – dotyczy to rozgałęzień bezwarunkowych oraz tych rozgałęzień warunkowych, których kod warunkowy jest znany i wskazuje na rozgałęzienie.
   (b) Czy rozgałęzienie nie nastąpi – dotyczy to rozgałęzień warunkowych, których kod warunkowy jest znany i wskazuje, że rozgałęzienie nie nastąpi.
   (c) Czy wynik nie może jeszcze być określony. W tym przypadku zakłada się wystąpienie rozgałęzienia w odniesieniu do rozgałęzień wstecznych (typowych dla pętli) oraz że rozgałęzienie nie nastąpi w odniesieniu do rozgałęzień ku przodowi. Rozkazy sekwencyjne następujące po rozgałęzieniu są kierowane do jednostek funkcjonalnych w trybie warunkowym. Gdy tylko wartość kodu warunkowego zostanie określona w jednostce wykonującej, jednostka rozgałęzień albo kasuje rozkazy w potoku i kontynuuje działanie z pobranym adresem docelowym w przypadku wystąpienia rozgałęzienia, albo sygnalizuje, że mają być wykonywane rozkazy warunkowe. Kompilator może użyć jednego bitu w kodzie rozkazu w celu odwrócenia tej „zaocznej" prognozy.

Strategia przewidywania rozgałęzień na podstawie historii została odrzucona, ponieważ zdaniem projektantów daje ona minimalne korzyści.

Jako przykład rezultatu przewidywania rozgałęzień rozważmy program z rys. 14.13 i załóżmy, że procesor rozgałęzień przewiduje niewystąpienie rozgałęzienia warunkowego (ocena „zaoczna" dotycząca rozgałęzienia ku przodowi). Na rysunku 14.14a pokazano wpływ tego przewidywania na potok, jeśli rzeczywiście rozgałęzienie nie nastąpiło. Podczas pierwszego cyklu do kolejki dyspozytorskiej jest ładowanych 8 rozkazów. Sześć pierwszych rozkazów to rozkazy dotyczące liczb całkowitych i są one wydawane do jednostki liczb całkowitych w tempie jednego rozkazu na cykl. Rozkaz rozgałęzienia warunkowego nie może być rozdysponowany, zanim nie dotrze do dolnej części kolejki dyspozytorskiej, co następuje w cyklu 5. Jednostka rozgałęzień przewiduje, że to rozgałęzienie nie nastąpi, wobec tego rozdysponowany warunkowo jest następny rozkaz w sekwencji (oznaczony jako D′). Rozgałęzienie nie może być rozstrzygnięte, zanim nie zostanie wykonany rozkaz porównania w cyklu 8. Wtedy właśnie procesor rozgałęzień potwierdza prawidłowość przewidywania i wykonywanie jest kontynuowane. Nie występują opóźnienia i potok pozostaje pełny.

```
if (a > 0)
        a = a + b + c + d + e;
else
        a = a - b - c - d - e;
```

(a) Program w języku C

```
                                #r1 wskazuje na a,
                                #r1+4 wskazuje na b,
                                #r1+8 wskazuje na c,
                                #r1+12 wskazuje na d,
                                #r1+16 wskazuje na e.
    lwz   r8=a(r1)              #ładuj a
    lwz   r12=b(r1,4)           #ładuj b
    lwz   r9=c(r1,8)            #ładuj c
    lwz   r10=d(r1,12)          #ładuj d
    lwz   r11=e(r1,16)          #ładuj e
    cmpi  cr0=r8,0              #porównaj natychmiastowe
    bc    ElSE/cr0/gt=false     #rozgałęziaj, jeśli nieprawdziwy
IF
    add   r12=r8,r12            #dodaj
    add   r12=r12,r9            #dodaj
    add   r12=r12,r10           #dodaj
    add   r4=r12,r11            #dodaj
    stw   a(r1)=r4              #zapisz
    b     OUT                   #rozgałęzienie bezwarunkowe
ELSE
    subf  r12=r12,r8            #odejmij
    subf  r12=r9,r12            #odejmij
    subf  r12=r10,r12           #odejmij
    subf  r4=r12,r11            #odejmij
    stw   a(r1)=r4              #zapisz
OUT:
```

(b) Program asemblerowy

Rysunek 14.13. Przykład programu z rozgałęzieniem warunkowym [WEIS94]

Zauważmy, że podczas cykli od 4 do 8 nie są pobierane żadne rozkazy. Dzieje się tak, ponieważ podczas tych cykli pamięć podręczna jest zajęta przez etapy dostępu do pamięci podręcznej dotyczące 5 rozkazów ładowania. Mimo to strumień rozkazów nie jest opóźniany, ponieważ kolejka dyspozytorska może zawierać 8 rozkazów.

Na rysunku 14.14b jest pokazany wpływ błędnego przewidywania rozgałęzienia (tzn. w rzeczywistości rozgałęzienie nastąpiło). W tym przypadku 3 rozkazy znajdujące się na etapie pobierania muszą być wyrzucone, a pobieranie jest wznawiane w odniesieniu do rozkazów umieszczonych po ELSE. W rezultacie etap wykonywania potoku jednostki liczb całkowitych pozostaje bezczynny podczas cykli 9 i 10, co oznacza stratę 2 cykli spowodowaną przez błędne przewidywanie.

```
                                1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
      lwz  r8=a(r1)             F  D  E  C  W
      lwz  r12=b(r1,4)          F  •  D  E  C  W
      lwz  r9=c(r1,8)           F  •  •  D  E  C  W
      lwz  r10=d(r1,12)         F  •  •  •  D  E  C  W
      lwz  r11=e(r1,16)         F  •  •  •  •  D  E  C  W
      cmpi cr0=r8,0             F  •  •  •  •  •  D  E
      bc   ELSE,cr0/gt=false    F  •  •  •  S
IF    add  r12=r8=r12           F  •  •  •  •  •  •  D' E  W
      add  r12=r12,r9                 F  •  •  •  •  •  D  E  W
      add  r12=r12,r10                F  •  •  •  •  •  •  D  E  W
      add  r4=r12,r11                                   F  •  D  E  W
      stw  a(r1)=r4                                     F  •  •  D  E  C
      b    OUT
ELSE: subf r12=r8,r12
      subf r12=r12,r9
      subf r12=r12,r10
      subf r4=r12,r11
      stw  a(r1)=r4
OUT:
```

(a) Przewidywanie prawidłowe: rozgałęzienie nie wystąpiło

```
                                1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
      lwz  r8=a(r1)             F  D  E  C  W
      lwz  r12=b(r1,4)          F  •  D  E  C  W
      lwz  r9=c(r1,8)           F  •  •  D  E  C  W
      lwz  r10=d(r1,12)         F  •  •  •  D  E  C  W
      lwz  r11=e(r1,16)         F  •  •  •  •  D  E  C  W
      cmpi cr0=r8,0             F  •  •  •  •  •  D  E
      bc   ELSE,cr0/gt=false    F  •  •  •  S
IF    add  r12=r8=r12           F  •  •  •  •  •  •  D'
      add  r12=r12,r9                 F  •  •  •  •  •
      add  r12=r12,r10                F  •  •  •  •  •
      add  r4=r12,r11
      stw  a(r1)=r4
      b    OUT
ELSE: subf r12=r8,r12                                   F  D  E  W
      subf r12=r12,r9                                   F  •  D  E  W
      subf r12=r12,r10                                  F  •  •  D  E  W
      subf r4=r12,r11                                   F  •  •  •  D  E  W
      stw  a(r1)=r4                                     F  •  •  •  •  D  E  C
OUT:
```

(b) Przewidywanie nieprawidłowe: rozgałęzienie wystąpiło

F – pobieranie
D – rozprowadzanie/dekodowanie
E – wykonywanie/adres
C – dostęp do pamięci podręcznej
W – zapis opóźniony
S – rozprowadzenie

Rysunek 14.14. Przewidywanie rozgałęzienia: nie wystąpi [WEIS94]

PowerPC 620

Procesor 620 jest pierwszą 64-bitową implementacją architektury PowerPC. Godną uwagi cechą tego wdrożenia jest występowanie sześciu niezależnych jednostek wykonawczych:

- jednostki rozkazów;
- trzech jednostek całkowitoliczbowych;
- jednostki ładowania/zapisu;
- jednostki zmiennopozycyjnej.

Organizacja ta umożliwia procesorowi jednoczesne rozdysponowywanie do 4 rozkazów przeznaczonych dla 3 jednostek liczb całkowitych i jednostki zmiennopozycyjnej.

W procesorze 620 zastosowano wydajną strategię przewidywania rozgałęzień obejmującą układy logiczne przewidywania, bufory przemianowania rejestrów i stacje rezerwowe wewnątrz jednostek wykonawczych. Gdy rozkaz jest pobierany, przypisuje się go do buforu przemianowania w celu tymczasowego przechowywania wyników rozkazu, takich jak zawartości rejestrów. Dzięki zastosowaniu buforów przemianowania, procesor może *wykonywać spekulatywnie* rozkazy oparte na przewidywaniu rozgałęzienia; jeśli przewidywanie okaże się błędne, to wyniki spekulatywnych rozkazów mogą być wyrzucone bez naruszania tablicy rejestrów. Gdy natomiast wynik rozgałęzienia jest potwierdzony, tymczasowe wyniki mogą być zapisane na stałe.

Każda jednostka ma dwie lub więcej stacji rezerwowych, w których są przechowywane rozkazy, które czekają na wyniki innych rozkazów. Rozwiązanie to umożliwia usunięcie tych rozkazów z jednostki rozkazów, co pozwala jej na kontynuowanie rozdysponowywania rozkazów do innych jednostek wykonawczych.

Procesor 620 może spekulatywnie wykonywać do 4 nierozstrzygniętych rozkazów rozgałęzień (wobec jednego w procesorze 601). Przewidywanie rozgałęzień jest oparte na wykorzystaniu tablicy historii rozgałęzień zawierającej 2048 zapisów. Symulacje przeprowadzone przez projektantów PowerPC wykazały, że wskaźnik celności przewidywań wynosi 90% [THOM94].

14.5. Polecana literatura

Praca [JOHN91] stanowi doskonałe książkowe ujęcie projektowania superskalarnego. Wartościowymi artykułami przeglądowymi na ten temat są [SMIT95] i [SIMA97]. W [JOUP89a] przeanalizowano paralelizm na poziomie rozkazu i różne metody maksymalizowania paralelizmu oraz porównano rozwiązania superpotokowe i superskalarne metodą symulacji. Dwoma najnowszymi artykułami, w których przedstawiono problemy podejścia superpotokowego, są [PATT01] i [MOSH01].

Artykuł [POPE91] zawiera dokładne omówienie proponowanej maszyny superskalarnej oraz doskonały wykład zagadnień projektowania związanych ze zmianą kolejności rozka-

zów. Inne spojrzenie na proponowany system znajduje się w [KUGA91]; w tym artykule rozpatrzono większość spośród najważniejszych zagadnień projektowania odnoszących się do implementacji superskalarnych. W [LEE91] przeanalizowano metody programowe, które mogą być użyte do zwiększania wydajności superskalarnej. [WALL91] stanowi interesujące studium możliwego zasięgu wykorzystywania równoległości na poziomie rozkazu w procesorze superskalarnym. W [LEE91] przeanalizowano metody programowe, jakie mogą być użyte do zwiększenia wydajności rozwiązań superskalarnych. [WALL91] to interesujące studium zakresu, w jakim paralelizm na poziomie rozkazu może być wykorzystany w procesorze superskalarnym.

W tomie I [INTE01a] znajduje się ogólny opis potoku Pentium 4; więcej szczegółów opublikowano w [INTE01b].

Szczegółową analizę przetwarzania potokowego rozkazów w PowerPC 601 zawiera [POTT94]. Dobre ujęcie tego tematu znajduje się również w [SHAN95].

HINT01 Hinton G. et al.: „The Microarchitecture of the Pentium 4 Processor". *Intel Technology Journal*, Q1 2001. http://developer.intel.com/technology/itj/

INTE01a Intel Corp.: *IA-32 Intel Architecture Software Developer's Manual (2 volumes)*. Document 245470 i 245471, Aurora, 2001.

INTE01b Intel Corp.: *Intel Pentium 4 Processor Optimization Reference Manual*. Document 248966-04, Aurora, 2001, CO, 2001 http://developer.intel.com/design/pentium4/manuals/248966.htm

JOHN91 Johnson M.: *Superscalar Microprocessor Design*. Englewood Cliffs, Prentice Hall, 1991.

JOUP89a Jouppi N., Wall D.: „Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines". *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.

KUGA91 Kuga M., Murakami K., Tomita S.: „DSNS (Dynamically-hazard resolved, Statistically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture". *Computer Architecture News*, June 1991.

LEE91 Lee R., Kwok A., Briggs F.: „The Floating Point Performance of a Superscalar SPARC Processor". *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

MOSH01 Moshovos A., Sohi G.: „Microarchitectural Innovations: Boosting Microprocessor Performance Beyond Semiconductor Technology Scaling". *Proceedings of the IEEE*, November 2001.

PATT01 Patt Y.: „Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution". *Proceedings of the IEEE*. November 2001.

POPE91 Popescu V. et al.: „The Metaflow Architecture". *IEEE Micro*, June 1991.

POTT94 Potter T. et al.: „Resolution of Data and Control-Flow Dependencies in the PowerPC 601". *IEEE Micro*, October 1994.

SHAN95 Shanley T.: *PowerPC System Architecture*. Reading, Addison-Wesley, 1995.

SIMA97 Sima D.: „Superscalar Instruction Issue". *IEEE Micro*, September/October 1997.

SMIT95 Smith J., Sohi G.: „The Microarchitecture of Superscalar Processors". *Proceedings of the IEEE*, December 1995.

WALL91 Wall D.: „Limits of Instruction-Level Parallelism". *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

14.6. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Antyzależność – *antidependency*
Kolejne kończenie rozkazów – *in-order completion*
Kolejne wydawanie rozkazów – *in-order issue*
Konflikt zasobów – *resource conflict*
Kończenie w zmienionej kolejności – *out-of-order completion*
Okno rozkazów – *instruction window*
Paralelizm na poziomie maszyny – *machine parallelism*
Paralelizm na poziomie rozkazu – *instruction-level-parallelism*
Przemianowywanie rejestrów – *register renaming*
Przetwarzanie superpotokowe – *superpipelined*
Przewidywanie rozgałęzień – *branch prediction*
Rozwiązanie superskalarne – *superscalar*
Rzeczywista zależność danych – *true data dependency*
Wydawanie rozkazów – *instruction issue*
Wydawanie w zmienionej kolejności – *out-of-order issue*
Zależność proceduralna – *procedural dependency*
Zależność wyjściowa – *output dependency*

Pytania kontrolne

14.1. Jaka jest podstawowa właściwość superskalarnego podejścia do projektowania procesorów?
14.2. Jaka jest różnica między podejściem skalarnym a superpotokowym?
14.3. Co to jest paralelizm na poziomie rozkazu?
14.4. Krótko zdefiniuj następujące terminy:
- Rzeczywista zależność danych
- Zależność proceduralna
- Konflikt zasobów
- Zależność wyjściowa
- Antyzależność

14.5. Czym różnią się od siebie paralelizm na poziomie rozkazu i paralelizm na poziomie maszyny?
14.6. Wymień i krótko zdefiniuj trzy rodzaje superskalarnej polityki wydawania rozkazów.
14.7. Do czego służy okno rozkazów?
14.8. Czym jest przemianowywanie rozkazu i do czego służy?
14.9. Jakie są podstawowe elementy superskalarnej organizacji procesora?

Problemy do rozwiązania

14.1. Gdy w procesorze superskalarnym jest stosowana zmieniona kolejność kończenia rozkazów, wznawianie wykonywania po przetworzeniu przerwania jest skomplikowane, ponieważ warunek wyjątkowy może być wykryty jako rozkaz, który wytworzył swój wynik w zmienionej kolejności. Program nie może być wznowiony począwszy od rozkazu następującego po rozkazie, który spowodował wystąpienie wyjątku, ponieważ następ-

ne rozkazy zostały już wykonane i byłyby wykonane po raz drugi. Zasugeruj mechanizm lub mechanizmy postępowania w takiej sytuacji.

14.2. Rozważ następującą sekwencję rozkazów, których składnia obejmuje kolejno kod operacji, rejestr docelowy i jeden lub dwa rejestry źródłowe:

```
0   ADD    R3, R1, R2
1   LOAD   R6, [R3]
2   AND    R7, R5, 3
3   ADD    R1, R6, R0
4   SRL    R7, R0, 8
5   OR     R2, R4, R7
6   SUB    R5, R3, R4
7   ADD    R0, R1, 10
8   LOAD   R6, [R5]
9   SUB    r2, r1, r6
10  AND    r3, r7, 15
```

Załóż użycie potoku 4-etapowego: pobranie, dekodowanie/wydanie, wykonanie, zapis. Załóż, że wszystkie etapy potoku zajmują jeden cykl z wyjątkiem etapu wykonania. W przypadku prostych, całkowitoliczbowych rozkazów arytmetycznych i logicznych etap wykonania zajmuje jeden cykl, jednak w przypadku ładowania z pamięci na etapie wykonania zużywanych jest 5 cykli.

Jeśli mamy prosty potok skalarny, w którym jednak dozwolona jest zmiana kolejności wykonywania, to możemy zbudować następującą tabelę wykonywania pierwszych siedmiu rozkazów:

Rozkaz	Pobieranie	Dekodowanie	Wykonanie	Zapis
0	0	1	2	3
1	1	2	4	9
2	2	3	5	6
3	3	4	10	11
4	4	5	6	7
5	5	6	8	10
6	6	7	9	12

Zapisy w kolumnach oznaczających kolejne cztery etapy potoku wskazują cykl zegara, przy którym każdy z rozkazów rozpoczyna daną fazę. W tym programie drugi rozkaz ADD (rozkaz 3) zależy od rozkazu LOAD (rozkaz 1) ze względu na jeden z argumentów, r6. Ponieważ rozkaz LOAD zajmuje 5 cykli zegara, a układy logiczne wydawania napotykają rozkaz zależny ADD po 2 cyklach zegara, układy te muszą opóźnić rozkaz ADD o 3 cykle zegara. Dysponując możliwością zmiany kolejności, procesor może zatrzymać rozkaz 3 w cyklu zegara 4, a następnie przejść do wydawania następnych trzech rozkazów niezależnych, których wykonywanie rozpocznie się w 6, 8 i 9 cyklu zegara. Zakończenie wykonywania LOAD nastąpi w cyklu 9, dzięki czemu rozkaz zależny ADD może być wprowadzony do wykonywania w cyklu 10.

(a) Uzupełnij powyższą tabelę.

(b) Zbuduj od nowa tabelę, zakładając że zmiana kolejności nie jest dozwolona. Jakie korzyści wynikają z możliwości stosowania zmiany kolejności?

(c) Zbuduj tabelę od nowa, zakładając rozwiązanie superskalarne, w którym na każdym etapie można przetwarzać 2 rozkazy.

14.3. W kolejce rozkazów jednostki dyspozytorskiej procesora PowerPC 601 rozkazy mogą być wydawane w zmienionej kolejności jednostkom przetwarzania rozgałęzień i zmiennopozycyjnej, jednak rozkazy przeznaczone dla jednostki całkowitoliczbowej muszą być wydawane tylko z dołu kolejki. Skąd to ograniczenie?

14.4. Wykonaj rysunek podobny do 14.14 dla następujących przypadków:

(a) przewidywanie rozgałęzienia: nastąpi; przewidywanie właściwe – rozgałęzienie nastąpiło;

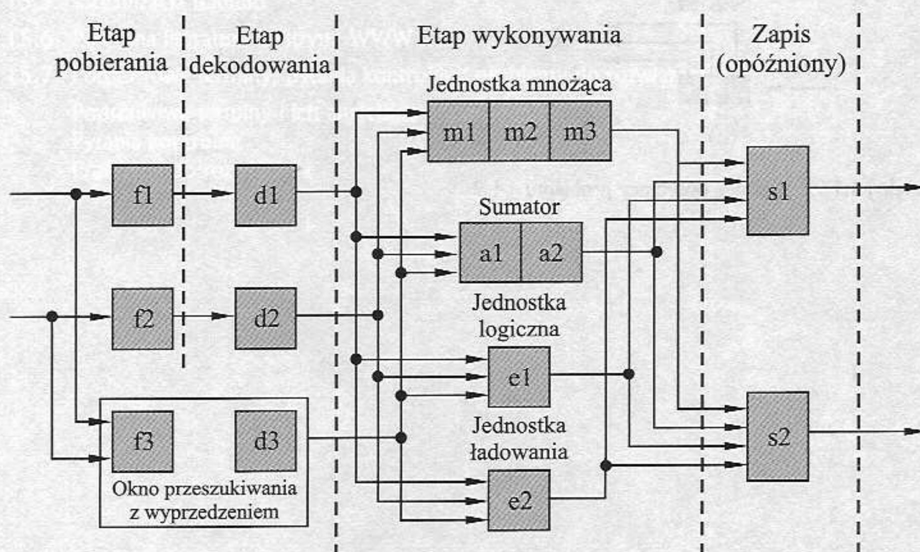
(b) przewidywanie rozgałęzienia: nastąpi; przewidywanie błędne – rozgałęzienie nie nastąpiło.

14.5. Rozważ następujący program w języku asemblerowym:

```
I1:  Move R3, R7          /R3 ← (R7)/
I2:  Load R8, (R3)        /R8 ← Pamięć (R3)/
I3:  Add R3, R3, 4        /R3 ← (R3) + 4/
I4:  Load R9, (R3)        /R9 ← Pamięć (R3)/
I5:  BLE R8, R9, L3       /Rozgałęziaj, jeśli (R9) > (R8)/
```

Program ten zawiera zależności zapis-zapis, odczyt-zapis i zapis-odczyt. Wskaż je.

14.6. Na rysunku 14.15 została pokazana organizacja procesora superskalarnego. Procesor ten może wydawać dwa rozkazy na cykl, jeśli nie ma konfliktu zasobów ani problemu zależności danych. Zasadniczo występują tu dwa potoki z czterema etapami przetwarzania (pobieranie, dekodowanie, wykonywanie i zapis). Każdy potok ma własną jednostkę pobierania, dekodowania i zapisu. Na etapie wykonywania są dostępne cztery jednostki funkcjonalne (mnożąca, sumująca, logiczna i ładowania), które są dynamicznie współużytkowane przez obydwa potoki. Dwie jednostki zapisu są dynamicznie współużytkowane przez dwa potoki, zależnie od dostępności w określonym cyklu. Istnieje też okno przeszukiwania wyprzedzającego z własnymi układami logicznymi pobierania i dekodowania. Okno to służy do wyprzedzającego przeszukiwania rozkazów pod kątem tych, które zostały wydane w zmienionej kolejności.



Rysunek 14.15. Dwupotokowy procesor superskalarny

Rozważ następujący program, który ma być wykonywany przez ten procesor:

```
I1:  Load R1, A        /R1 ← Pamięć (A)/
I2:  Add R2, R1        /R2 ← (R2) + R(1)/
I3:  Add R3, R4        /R3 ← (R3) + R(4)/
I4:  Mul R4, R5        /R4 ← (R4) + R(5)/
I5:  Comp R6           /R6 ← (R6)/
I6:  Mul R6, R7        /R3 ← (R3) + R(4)/
```

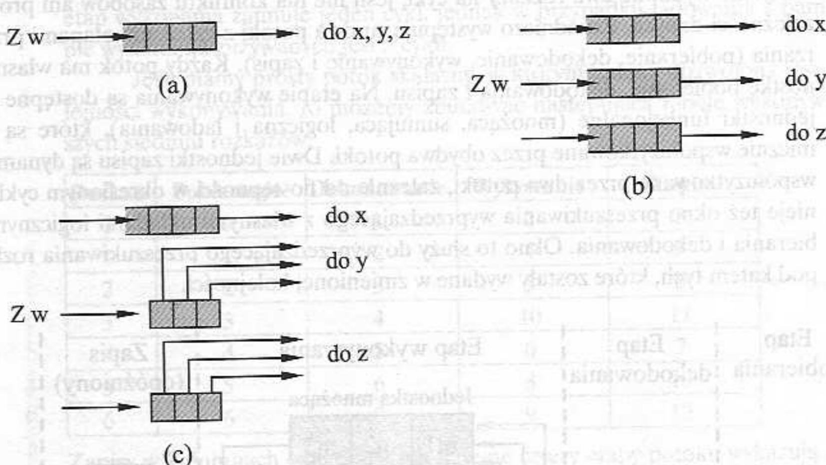
(a) Jakie zależności występują w tym programie?

(b) Zademonstruj potokowe działanie tego programu w procesorze z rys. 14.15, posługując się polityką kolejnego wydawania i kolejnego kończenia i wykorzystując prezentację podobną do pokazanej na rys. 14.2.

(c) Powtórz to samo w odniesieniu do kolejnego wydawania i kończenia w zmienionej kolejności.

(d) Powtórz to samo dla wydawania w zmienionej kolejności i kończenia w zmienionej kolejności.

14.7. Rysunek 14.16 pochodzi z pewnego artykułu poświęconego projektowaniu superskalarnemu. Objaśnij trzy części tego rysunku oraz zdefiniuj w, x, y i z.



Rysunek 14.16. Rysunek dotyczący problemu 14.7

Rozdział 15

Architektura IA-64

15.1. Uzasadnienie

15.2. Organizacja ogólna

15.3. Predykacja, spekulacja i potokowanie programowe

Format rozkazu
Format języka asemblerowego
Wykonywanie predykatywne
Spekulacja sterowaniem
Spekulacja danymi
Potokowanie programowe

15.4. Architektura listy rozkazów IA-64

Stos rejestrów
Znacznik bieżącej ramki i stan poprzedniej funkcji

15.5. Organizacja Itanium

15.6. Polecana literatura i witryny WWW

15.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

PODSTAWOWE SPOSTRZEŻENIA

- Architektura listy rozkazów IA-64 stanowi nowe podejście do zapewnienia silnego wsparcia paralelizmu na poziomie rozkazu. Różni się ona znacznie od rozwiązań stosowanych w architekturach superskalarnych.
- Najbardziej godnymi uwagi właściwościami architektury IA-64 są: sprzętowe wsparcie wykonywania predykatywnego, spekulacja sterowaniem, spekulacja danymi i potokowanie programowe.
- Przy **wykonywaniu predykatywnym** każdy rozkaz IA-64 obejmuje odniesienie do 1-bitowego rejestru predykacji i jest wykonywany jedynie wówczas, gdy wartość predykacji wynosi 1 (prawda). Umożliwia to procesorowi spekulatywne wykonywanie zarówno rozgałęzień, jak i instrukcji IF (jeżeli), oraz odsyłanie rozkazów tylko po określeniu warunku.
- Przy **spekulacji sterowaniem** rozkaz ładowania jest przenoszony do wcześniejszej części programu, a w jego początkowej pozycji umieszcza się rozkaz sprawdzenia. Wczesne ładowanie oszczędza czas cyklu; jeśli ładowanie prowadzi do wyjątku, wyjątek ten nie jest aktywowany, dopóki rozkaz sprawdzenia nie określi, czy ładowanie powinno nastąpić.
- Przy **spekulacji danymi** ładowanie jest przenoszone przed rozkaz zapisu, który mógłby zmienić lokację w pamięci będącą źródłem ładowania. Dokonuje się następnie sprawdzenia w celu upewnienia się, czy ładowanie odbiera właściwą wartość pamięci.
- **Potokowanie programowe** jest metodą polegającą na umożliwieniu równoległego wykonywania rozkazów pochodzących z wielu iteracji w ramach pętli.

Wraz z Pentium 4 rodzina mikroprocesorów zapoczątkowana przez 8086 i będąca najbardziej udanym szeregiem produktów techniki komputerowej wydaje się dobiegać końca. Intel połączył wysiłki z firmą Hewlett-Packard (HP) w celu opracowania nowej architektury 64-bitowej, zwanej IA-64. Nie jest ona 64-bitowym rozszerzeniem 32-bitowej architektury Intel x86 ani też adaptacją 64-bitowej architektury Hewlett-Packard, znanej jako PA-RISC. IA-64 jest nową architekturą opartą na latach badań prowadzonych w obydwu firmach i na uniwersytetach. Wykorzystuje się w niej rozbudowane układy i duże szybkości dostępne w nowych generacjach mikroukładów, systematycznie korzystając z paralelizmu. Architektura IA-64 stanowi znaczne odstępstwo od ogólnej tendencji do stosowania rozwiązań superskalarnych, która zdominowała dotychczasowy rozwój procesorów.

Rozpoczniemy ten rozdział od omówienia czynników uzasadniających pojawienie się tej nowej architektury. Następnie zapoznamy się z ogólną organizacją wspomagającą tę architekturę. Przeanalizujemy dość szczegółowo podstawowe właściwości architektury IA-64 sprzyjające paralelizmowi na poziomie rozkazu. Na zakończenie poznamy architekturę listy rozkazów IA-64 i organizację Itanium.

15.1. Uzasadnienie

Podstawowe koncepcje leżące u podstaw IA-64 są następujące:

- Paralelizm na poziomie rozkazu widoczny raczej w rozkazach maszynowych, niż wyznaczany przez procesor podczas pracy.
- Długie i bardzo długie słowa rozkazów (LIW, VLIW).
- Predykacja rozgałęzień (*nie* to samo co predykcja, czyli przewidywanie).
- Spekulatywne ładowanie.

Intel i HP określają tę kombinację koncepcji jako **jawnie równoległe przetwarzanie rozkazów** (*explicitly parallel instruction computing* – EPIC). Obie firmy mianem **EPIC** określają technologię lub zbiór technik. **IA-64** jest architekturą listy rozkazów przeznaczonej do implementacji technologii EPIC. Pierwszy produkt Intel oparty na IA-64 jest określany jako **Itanium.** Spodziewane są następne produkty oparte na tej samej architekturze.

W tabeli 15.1 zostały zestawione podstawowe różnice między IA-64 a tradycyjnym podejściem superskalarnym.

Tabela 15.1. Tradycyjne podejście superskalarne a architektura IA-64

Rozwiązanie superskalarne	IA-64
Rozkazy typu RISC, jeden rozkaz na słowo	Rozkazy typu RISC pakietowane po trzy
Wiele równoległych jednostek wykonujących	Wiele równoległych jednostek wykonujących
Zmiana kolejności i optymalizacja strumienia rozkazów podczas realizowania programu	Zmiana kolejności i optymalizacja strumienia rozkazów podczas kompilowania
Przewidywanie rozgałęzień ze spekulatywnym wykonywaniem jednej ścieżki	Spekulatywne wykonywanie obydwu ścieżek rozgałęzienia
Ładowanie danych z pamięci tylko wówczas, gdy są potrzebne. Najpierw próba znalezienia danych w pamięci podręcznej	Spekulatywne ładowanie danych, zanim są potrzebne. Najpierw próba znalezienia danych w pamięci podręcznej

Dla firmy Intel przestawienie się na nową architekturę, która nie jest zgodna sprzętowo z architekturą rozkazów x86, było decyzją doniosłą. Wynikła ona jednak z dyktatu technologii. Gdy zapoczątkowano rodzinę x86 pod koniec lat siedemdziesiątych, mikroukład procesora zawierał dziesiątki tysięcy tranzystorów i był w zasadzie urządzeniem skalarnym. Oznacza to, że rozkazy były przetwarzane kolejno, z niewielkim udziałem potoku lub bez przetwarzania potokowego. Gdy w połowie lat osiemdziesiątych liczba tranzystorów wzrosła do setek tysięcy, Intel wprowadził przetwarzanie potokowe (np. rys. 12.18). Tymczasem inni wytwórcy dążyli do wykorzystania zwiększonej liczby tranzystorów i szybkości, wprowadzając rozwiązanie RISC, które umożliwiało skuteczniejsze przetwarzanie potokowe, a w późniejszym czasie kombinację RISC i rozwiązań superskalarnych, obejmującą wiele jednostek wykonujących. Wraz z Pentium Intel poczynił umiarkowane starania, aby zastosować metody superskalarne, umożliwiając jednoczesne wykonywanie dwóch rozka-

zów CISC. Następnie w Pentium Pro, Pentium II i Pentium 4 wprowadzono odwzorowanie rozkazów CISC na mikrooperacje w stylu RISC oraz śmielsze zastosowanie technik superskalarnych. Podejście to pozwoliło na efektywne wykorzystywanie mikroukładów zawierających miliony tranzystorów. Jeśli jednak chodzi o procesory następnej generacji, poza Pentium, Intel i inni wytwórcy stanęli przed koniecznością efektywnego wykorzystania dziesiątków milionów tranzystorów w jednym mikroukładzie procesora.

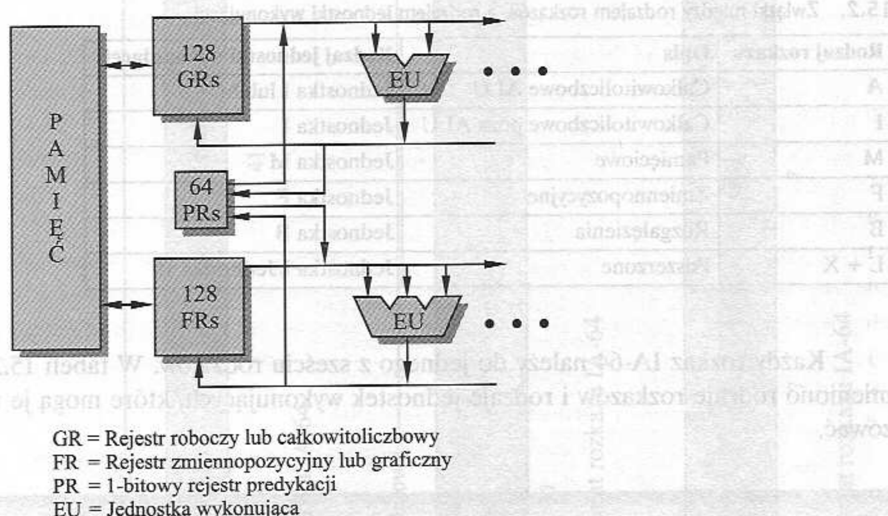
Projektanci procesorów mają niewielki wybór, jeśli chodzi o wykorzystanie tego nadmiaru tranzystorów. Jedno z rozwiązań to upchnięcie tych dodatkowych tranzystorów w większych pamięciach podręcznych. Do pewnego stopnia może to zwiększać wydajność, lecz w końcu korzyści zaczynają maleć i zwiększenie pamięci podręcznej prowadzi do nieznacznej tylko poprawy współczynnika trafień. Inna możliwość to zwiększanie stopnia superskalarności przez wprowadzenie większej liczby jednostek wykonujących. W tym przypadku szybko narasta złożoność procesora. W miarę dodawania coraz większej liczby jednostek wykonujących („poszerzania" procesora) wymagane są coraz bardziej rozbudowane układy logiczne sterujące ich współdziałaniem. Konieczne jest udoskonalenie przewidywania rozgałęzień, musi być zastosowane przetwarzanie w zmienionej kolejności i trzeba użyć dłuższych potoków. Z kolei przy większej liczbie coraz dłuższych potoków konsekwencje błędnego przewidywania są coraz dotkliwsze. Wykonywanie rozkazów w zmienionej kolejności wymaga wielkiej liczby przemianowywanych rejestrów i złożonych układów kontrolowania zależności. W rezultacie najlepsze współczesne procesory są w stanie odsyłać co najwyżej 6 rozkazów na cykl, a zwykle jest ich mniej.

Aby stawić czoła tym problemom, Intel i HP zaproponowały ogólne rozwiązanie umożliwiające efektywne użycie procesora z wieloma równoległymi jednostkami wykonującymi. Istotą tego nowego podejścia jest koncepcja jawnego paralelizmu. Polega ona na tym, że kompilator statycznie szereguje rozkazy podczas kompilowania, a procesor nie musi szeregować ich dynamicznie podczas pracy. Kompilator określa, które rozkazy mogą być wykonywane równolegle i włącza tę informację do rozkazów maszynowych. Procesor wykorzystuje ją do równoległego wykonywania rozkazów. Jedną z korzyści płynących z takiego rozwiązania jest to, że procesor EPIC nie potrzebuje tak złożonych układów, jak procesor superskalarny przetwarzający rozkazy w zmienionej kolejności. Ponadto, podczas gdy procesor ma tylko nanosekundy na określenie możliwości przetwarzania równoległego, kompilator dysponuje czasem o rzędy wielkości dłuższym, aby bez pośpiechu przeanalizować kod i widzieć program jako całość.

15.2. Organizacja ogólna

Jak każda architektura procesora, rozwiązanie IA-64 może być implementowane w postaci wielu różnych organizacji. Na rysunku 15.1 zasugerowano ogólną organizację maszyny IA-64. Oto jej podstawowe właściwości:

- **Wielka liczba rejestrów.** Z formatu rozkazu IA-64 wynika użycie 256 rejestrów: 128 64-bitowych rejestrów całkowitoliczbowych, logicznych i roboczych oraz 128 82-bitowych rejestrów zmiennopozycyjnych i graficznych. Istnieją również 64 1-bitowe rejestry predykacji używane przy predykatywne wykonywaniu rozkazów, co zostanie objaśnione w dalszym ciągu.
- **Wiele jednostek wykonywania.** Typowa dzisiejsza maszyna superskalarna może obsługiwać cztery równoległe potoki, posługując się czterema równoległymi jednostkami wykonywania zarówno w całkowitoliczbowej, jak i w zmiennopozycyjnej części procesora. Oczekuje się, że architektura IA-64 będzie wdrażana w systemach z co najmniej ośmioma jednostkami równoległymi.



Rysunek 15.1. Ogólna organizacja odpowiadająca architekturze IA-64

Tablica rejestrów jest całkiem duża w porównaniu z większością maszyn RISC i superskalarnych. Wynika to z tego, że wielka liczba rejestrów jest wymagana do obsługi wysokiego stopnia paralelizmu. W tradycyjnej maszynie superskalarnej język maszynowy (i asemblerowy) korzysta z niewielkiej liczby rejestrów widzialnych, procesor zaś odwzorowuje je na większej liczbie rejestrów, posługując się metodą przemianowywania rejestrów i analizą zależności. Ponieważ zależy nam na ujawnieniu paralelizmu i uwolnieniu procesora od ciężaru przemianowywania i analizy zależności, potrzebujemy wielkiej liczby rejestrów jawnych.

Liczba jednostek wykonujących jest funkcją liczby tranzystorów dostępnych w określonej implementacji. Procesor będzie korzystał z paralelizmu w takim stopniu, w jakim będzie w stanie to czynić. Jeśli na przykład strumień rozkazów w języku maszynowym wskazuje, że osiem rozkazów całkowitoliczbowych może być wykonywanych równolegle, procesor dysponujący czterema potokami całkowitoliczbowymi wykona je w dwóch częściach. Natomiast procesor z ośmioma potokami wykona wszystkie osiem rozkazów jednocześnie.

W architekturze IA-64 zostały zdefiniowane cztery rodzaje jednostek wykonujących:

- **Jednostka I.** Przeznaczona dla całkowitoliczbowych operacji arytmetycznych, przesunięć połączonych z dodawaniem, operacji logicznych, porównywania oraz całkowitoliczbowych rozkazów multimedialnych.
- **Jednostka M.** Prowadzi operacje ładowania i zapisu między rejestrem a pamięcią oraz pewne całkowitoliczbowe operacje ALU.
- **Jednostka B.** Rozkazy rozgałęziania.
- **Jednostka F.** Rozkazy zmiennopozycyjne.

Tabela 15.2. Związki między rodzajem rozkazów a rodzajem jednostki wykonującej

Rodzaj rozkazu	Opis	Rodzaj jednostki wykonującej
A	Całkowitoliczbowe ALU	Jednostka I lub M
I	Całkowitoliczbowe poza ALU	Jednostka I
M	Pamięciowe	Jednostka M
F	Zmiennopozycyjne	Jednostka F
B	Rozgałęzienia	Jednostka B
L + X	Poszerzone	Jednostka I/Jednostka B

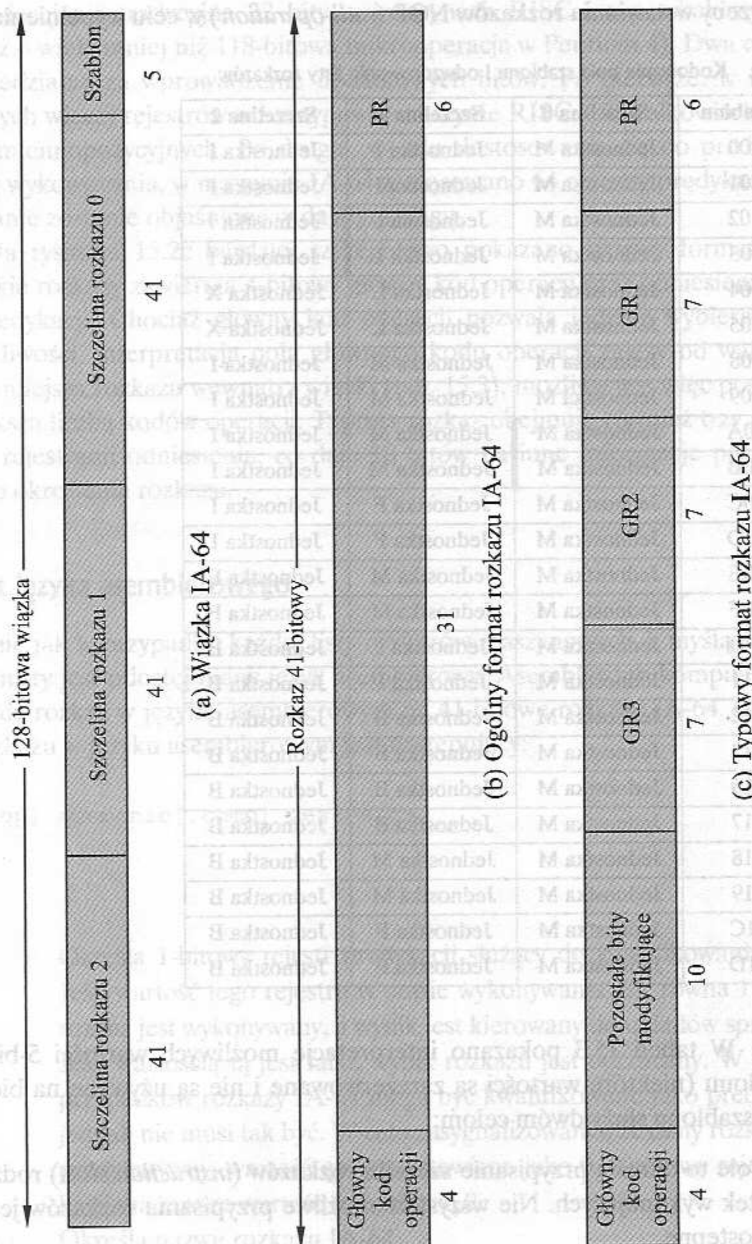
Każdy rozkaz IA-64 należy do jednego z sześciu rodzajów. W tabeli 15.2 wymieniono rodzaje rozkazów i rodzaje jednostek wykonujących, które mogą je realizować.

15.3. Predykacja, spekulacja i potokowanie programowe

W tym podrozdziale zapoznamy się z podstawowymi właściwościami architektury IA-64, które wspierają paralelizm na poziomie rozkazu. Najpierw dokonamy przeglądu formatu rozkazu IA-64 oraz – aby stworzyć podstawy dla przytaczanych w tym podrozdziale przykładów – zdefiniujemy ogólny format rozkazów języka asemblerowego IA-64.

Format rozkazu

W IA-64 jest zdefiniowana 128-bitowa **wiązka** (*bundle*) zawierająca trzy rozkazy zwane **sylabami,** a także pole szablonu (rys. 15.2a). Procesor może pobierać rozkazy jednocześnie po jednej lub po kilka wiązek; pobranie jednej wiązki oznacza doprowadzenie trzech rozkazów. Pole szablonu zawiera informacje wskazujące, które rozkazy mogą być wykonywane równolegle. Interpretacja pola szablonu nie jest ograniczona do jednej wiązki. Procesor może przeglądać wiele wiązek w celu stwierdzenia, które rozkazy mogą być wykonywane równolegle. Na przykład strumień rozkazów może być taki, że równolegle może być wykonywanych osiem rozkazów. Kompilator



PR = Rejestr predykacji
GR = Rejestr roboczy lub zmiennopozycyjny

Rysunek 15.2. Format rozkazu IA-64

zmieni kolejność rozkazów w taki sposób, że te osiem rozkazów znajdzie się w sąsiadujących ze sobą wiązkach i ustawi bity szablonu tak, aby procesor „wiedział", że są to rozkazy niezależne.

Rozkazy połączone w wiązki nie muszą być uporządkowane w kolejności zgodnej z programem początkowym. Co więcej, dzięki elastyczności pola szablonu kompilator może mieszać ze sobą w jednej wiązce rozkazy zależne i niezależne. W przeciwieństwie do niektórych poprzednich rozwiązań VLIW, w IA-64 nie ma potrzeby wstawiania rozkazów NOP (*null-operation*) w celu wypełnienia wiązek.

Tabela 15.3. Kodowanie pola szablonu i odwzorowanie listy rozkazów

Szablon	Szczelina 0	Szczelina 1	Szczelina 2
00	Jednostka M	Jednostka I	Jednostka I
01	Jednostka M	Jednostka I	Jednostka I
02	Jednostka M	Jednostka I	Jednostka I
03	Jednostka M	Jednostka I	Jednostka I
04	Jednostka M	Jednostka L	Jednostka X
05	Jednostka M	Jednostka L	Jednostka X
08	Jednostka M	Jednostka M	Jednostka I
09	Jednostka M	Jednostka M	Jednostka I
0A	Jednostka M	Jednostka M	Jednostka I
0B	Jednostka M	Jednostka M	Jednostka I
0C	Jednostka M	Jednostka F	Jednostka I
0D	Jednostka M	Jednostka F	Jednostka I
0E	Jednostka M	Jednostka M	Jednostka F
0F	Jednostka M	Jednostka M	Jednostka F
10	Jednostka M	Jednostka I	Jednostka B
11	Jednostka M	Jednostka I	Jednostka B
12	Jednostka M	Jednostka B	Jednostka B
13	Jednostka M	Jednostka B	Jednostka B
16	Jednostka M	Jednostka B	Jednostka B
17	Jednostka M	Jednostka B	Jednostka B
18	Jednostka M	Jednostka M	Jednostka B
19	Jednostka M	Jednostka M	Jednostka B
1C	Jednostka M	Jednostka F	Jednostka B
1D	Jednostka M	Jednostka F	Jednostka B

W tabeli 15.3 pokazano interpretację możliwych wartości 5-bitowego pola szablonu (niektóre wartości są zarezerwowane i nie są używane na bieżąco). Wartość szablonu służy dwóm celom:

1. Pole to określa przypisanie **szczelin rozkazów** (*instruction slots*) rodzajom jednostek wykonujących. Nie wszystkie możliwe przypisania rozkazów jednostkom są dostępne.

2. Pole to wskazuje obecność jakichkolwiek **zatrzymań** (*stops*). Zatrzymanie wskazuje sprzętowi, że jeden lub wiele rozkazów znajdujących się przed zatrzymaniem może wykazywać pewne rodzaje zależności od zasobów w stosunku do jednego lub wielu rozkazów znajdujących się po zatrzymaniu. W tabeli zatrzymanie jest wskazane za pomocą grubej linii pionowej.

Każdy rozkaz ma 41-bitowy format o ustalonej długości (rys. 15.2b). Jest to nieco więcej niż tradycyjne 32 bity w maszynach RISC i superskalarnych RISC (chociaż o wiele mniej niż 118-bitowa mikrooperacja w Pentium 4). Dwa czynniki są odpowiedzialne za wprowadzenie dodatkowych bitów. Po pierwsze, w IA-64 jest używanych więcej rejestrów niż w typowej maszynie RISC: 128 całkowitoliczbowych i 128 zmiennopozycyjnych. Po drugie, w celu dostosowania się do predykatywnej metody wykonywania, w maszynie IA-64 zastosowano 64 rejestry predykacji. Ich zastosowanie zostanie objaśnione w dalszym ciągu.

Na rysunku 15.2c bardziej szczegółowo pokazano typowy format rozkazu. Wszystkie rozkazy zawierają 4-bitowy główny kod operacji oraz odniesienie do rejestru predykacji. Chociaż główny kod operacji pozwala jedynie wybierać spośród 16 możliwości, interpretacja pola głównego kodu operacji zależy od wartości szablonu i miejsca rozkazu wewnątrz wiązki (tab. 15.3); możliwe jest więc posługiwanie się większą liczbą kodów operacji. Typowy rozkaz obejmuje również trzy pola związane z rejestrami odniesienia, co daje 10 bitów na inne informacje potrzebne do pełnego określenia rozkazu.

Format języka asemblerowego

Podobnie jak w przypadku każdej listy rozkazów maszynowych, z myślą o wygodzie programisty jest udostępniany język asemblerowy. Asembler lub kompilator tłumaczy każdy rozkaz w języku asemblerowym na 41-bitowe rozkazy IA-64. Ogólny format rozkazu w języku asemblerowym jest następujący:

```
[qp] mnemonic[.comp] dest=srcs
```

gdzie

`qp` Określa 1-bitowy rejestr predykacji służący do kwalifikowania rozkazu. Jeśli wartość tego rejestru w czasie wykonywania jest równa 1 (prawda), rozkaz jest wykonywany, a wynik jest kierowany do układów sprzętowych. Jeśli wartością tą jest fałsz, wynik rozkazu jest odrzucany. W większości przypadków rozkazy IA-64 mogą być kwalifikowane jako predykatywne, jednak nie musi tak być. W celu zasygnalizowania, że dany rozkaz nie jest predykatywny, wartość qp jest ustawiana jako 0, a zerowy rejestr predykacji ma zawsze wartość stałą równą 0.

`mnemonic` Określa nazwę rozkazu IA-64.

`comp` Określa jeden lub więcej pól uzupełniania rozkazu, oddzielonych od siebie kropkami i służących do kwalifikowania mnemonika. Nie wszystkie rozkazy wymagają stosowania pól uzupełniających.

`dest` Określa jeden lub wiele argumentów docelowych, przy czym typowym przypadkiem jest jeden argument.

`srcs` Określa jeden lub wiele argumentów źródłowych. Większość rozkazów dotyczy dwóch lub większej liczby argumentów.

W każdym wierszu wszelkie znaki znajdujące się na prawo od podwójnego ukośnika „//" są traktowane jako komentarz. Grupy rozkazów i zatrzymania są wskazywane za pomocą podwójnego średnika ";;". Grupa rozkazów jest definiowana jako sekwencja rozkazów nie wykazujących zależności typu „odczyt po zapisie" lub „zapis po zapisie". Procesor może je wydawać bez sprzętowego sprawdzania zależności rejestrowych. Oto prosty przykład:

```
ld8 r1 = [r5] ;;        // Pierwsza grupa
add r3 = r1, r4         // Druga grupa
```

Pierwszy rozkaz powoduje wczytanie wartości 8-bajtowej z lokacji pamięci, której adres znajduje się w rejestrze r5, po czym umieszcza tę wartość w rejestrze r1. Drugi rozkaz powoduje dodanie zawartości r1 i r4 oraz umieszczenie wyniku w r3. Ponieważ drugi rozkaz zależy od wartości znajdującej się w r1, która ulega zmianie podczas wykonywania pierwszego rozkazu, obydwa te rozkazy nie mogą się znajdować w tej samej grupie przeznaczonej do równoległego wykonywania.

Oto przykład bardziej złożony, z wieloma zależnościami rejestrowymi:

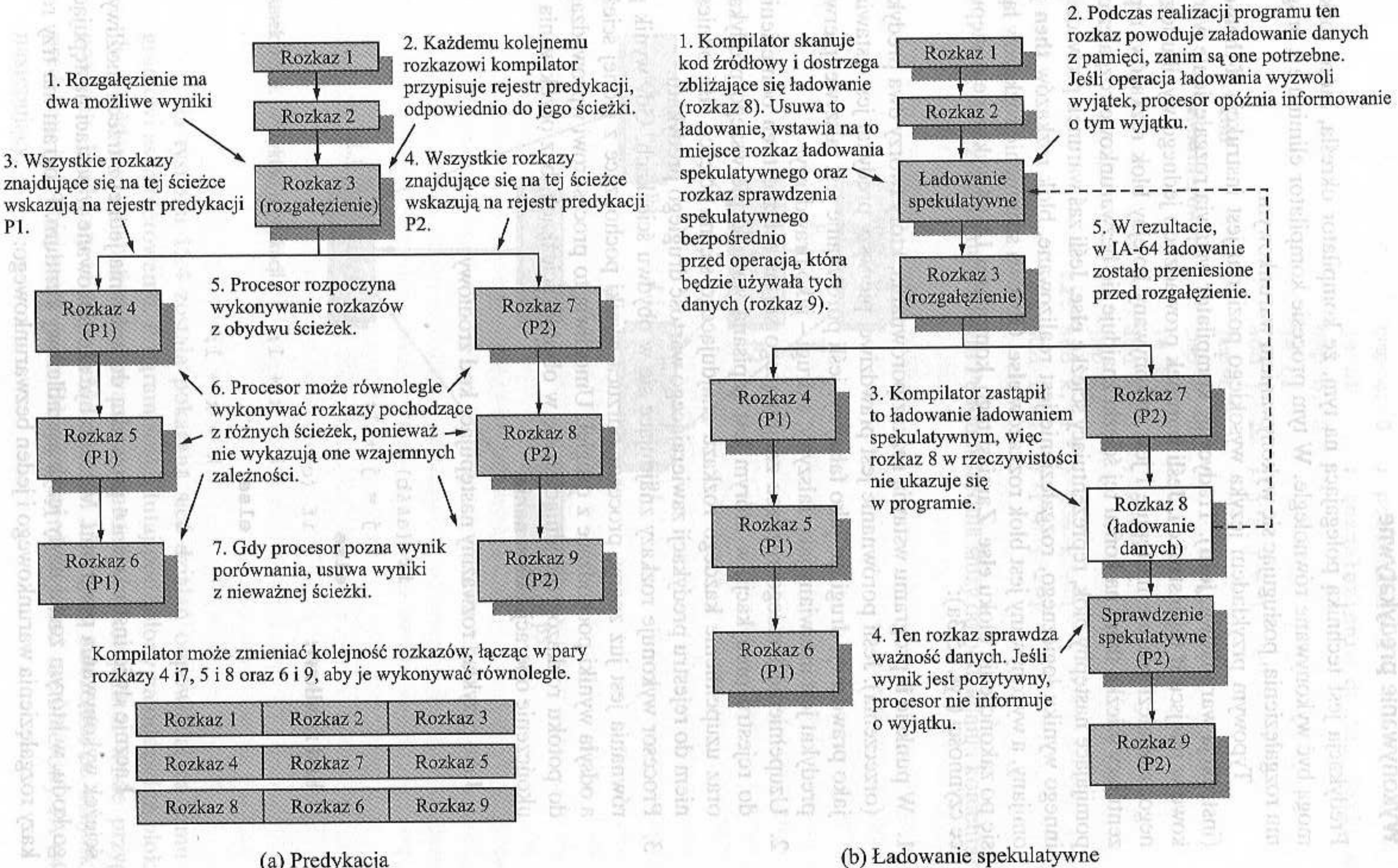
```
ld8 r1 = [r5]           // Pierwsza grupa
sub r6 = r8, r9 ;;      // Pierwsza grupa
add r3 = r1, r4         // Druga grupa
st8 [r6] = r12          // Druga grupa
```

Ostatni rozkaz powoduje zapisanie zawartości r12 w lokacji pamięci, której adres znajduje się w r6.

Jesteśmy już przygotowani do tego, aby zapoznać się z czterema podstawowymi mechanizmami w architekturze IA-64 mającymi na celu wspieranie paralelizmu na poziomie rozkazu:

- Predykacja
- Spekulacja sterowaniem
- Spekulacja danymi
- Potokowanie programowe

Na rysunku 15.3, opartym na [HALF97], zademonstrowano pierwsze dwie spośród tych metod, które zostaną przedyskutowane w tym i w następnym podrozdziale.



Rysunek 15.3. Predykacja i ładowanie spekulatywne w IA-64

Wykonywanie predykatywne

Predykacja jest techniką polegającą na tym, że kompilator określa, które rozkazy mogą być wykonywane równolegle. W tym procesie kompilator eliminuje z programu rozgałęzienia, posługując się wykonywaniem warunkowym.

Typowym przykładem języka wysokiego poziomu jest instrukcja **if-then-else** (instrukcja warunkowa, **jeśli**). Tradycyjny kompilator wstawia rozgałęzienie warunkowe w miejscu **if** tej instrukcji. Jeśli warunek prowadzi do jednego wyniku logicznego, rozgałęzienie nie następuje i jest wykonywany kolejny blok rozkazów, reprezentujący ścieżkę **then**; na końcu tej ścieżki znajduje się bezwarunkowe rozgałęzienie pomijające następny blok, reprezentujący ścieżkę **else**. Jeśli zaś warunek prowadzi do innego wyniku logicznego, rozgałęzienie jest realizowane, blok rozkazów **then** jest omijany, a wykonywany jest blok rozkazów **else**. Obydwa strumienie rozkazów łączą się po zakończeniu bloku **else**. Zamiast tego, kompilator IA-64 wykonuje następujące czynności (rys. 15.3a):

1. W punkcie **if** programu wstawia rozkaz porównania, który tworzy dwa predykaty (orzeczenia). Jeśli porównanie jest prawdziwe, pierwszy predykat jest ustawiany jako prawdziwy, drugi zaś jako fałszywy; jeśli porównanie jest fałszywe, pierwszy predykat jest ustawiany jako fałszywy, a drugi – jako prawdziwy.
2. Uzupełnienie każdego rozkazu znajdującego się na ścieżce **then** odniesieniem do rejestru predykacji, w którym jest zapisana wartość pierwszego predykatu, oraz uzupełnienie każdego rozkazu znajdującego się na ścieżce **else** odniesieniem do rejestru predykacji zawierającego wartość drugiego predykatu.
3. Procesor wykonuje rozkazy znajdujące się w obydwu ścieżkach. Gdy wynik porównania jest już znany, procesor odrzuca wyniki pochodzące z jednej ścieżki, a odsyła wyniki pochodzące z drugiej. Umożliwia to procesorowi wprowadzanie do potoku rozkazów znajdujących się w obydwu ścieżkach, bez oczekiwania na ukończenie operacji porównania.

Jako przykład rozważmy następujący kod źródłowy:

Kod źródłowy:

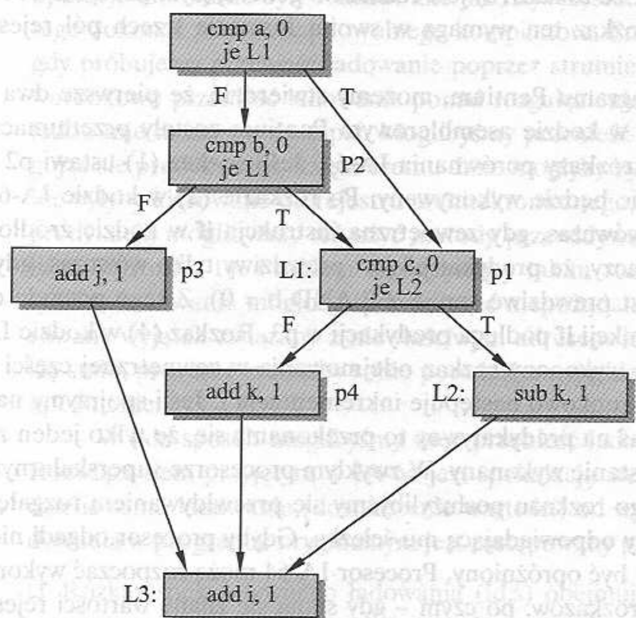
```
if (a&&b)
    j = j + 1;
else
    if (c)
        k = k + 1;
    else
        k = k - 1;
i = i + 1 ;
```

Łącznie dwie instrukcje `if` prowadzą do wybrania jednej z trzech możliwych ścieżek wykonywania programu. Może to być skompilowane w postaci następującego kodu, w którym został użyty język asemblerowy Pentium. Program ma trzy rozkazy rozgałęzienia warunkowego i jeden bezwarunkowego:

Kod asemblerowy:

```
    cmp a, 0      ; porównanie a z 0
    je L1         ; rozgałęzienie do L1, jeśli a = 0
    cmp b, 0
    je L1
    add j, 1      ; j = j + 1
    jmp L3
L1: cmp c, 0
    je L2
    add k, 1      ; k = k + 1
    jmp L3
L2: sub k, 1      ; k = k - 1
L3: add i, 1      ; i = i + 1
```

W języku asemblerowym Pentium średnik służy do oddzielania komentarzy.



Rysunek 15.4. Przykład predykacji

Na rysunku 15.4 została pokazana sieć działań odpowiadająca temu programowi asemblerowemu. Program został tutaj podzielony na oddzielne bloki kodu. Każdemu blokowi wykonywanemu warunkowo kompilator może przypisać predykat. Predykaty te są pokazane na rys. 15.4. Przy założeniu, że wszystkie te predykaty były inicjowane jako fałszywe, wynikowy kod asemblerowy IA-64 jest następujący:

Kod predykatywny:

```
(1)          cmp.eq p1, p2 = 0, a ;;
(2)   (p2)   cmp.eq p1, p3 = 0, b
(3)   (p3)   add j = 1, j
(4)   (p1)   cmp.ne p4, p5 = 0, c
(5)   (p4)   add k = 1, k
(6)   (p5)   add k = -1, k
(7)          add i = 1, i
```

Rozkaz (1) porównuje zawartość rejestru symbolicznego z 0; ustawia wartość rejestru predykacji p1 jako 1 (prawda) i p2 jako 0 (fałsz), jeśli relacja jest prawdziwa. Jeśli natomiast jest ona fałszywa, ustawia wartość predykatu p1 jako 0, a p2 – jako 1. Rozkaz (2) ma być wykonywany jedynie wówczas, gdy predykat p2 jest prawdziwy (tzn. jeśli prawdziwe jest a, co jest równoważne a ≠ 0). Procesor pobierze, zdekoduje i zacznie wykonywać ten rozkaz, jednak podejmie decyzję na temat odesłania wyniku dopiero po tym, gdy stwierdzi, czy wartość predykatu p1 jest równa 1, czy 0. Zauważmy, że rozkaz (2) jest rozkazem generującym predykat i że sam podlega predykacji. Rozkaz ten wymaga w swoim formacie trzech pól rejestrów predykacji.

Wracając do programu Pentium, możemy stwierdzić, że pierwsze dwa rozgałęzienia warunkowe w kodzie asemblerowym Pentium zostały przetłumaczone na dwa predykatywne rozkazy porównania IA-64. Jeśli rozkaz (1) ustawi p2 jako fałszywy, rozkaz (2) nie będzie wykonywany. Po rozkazie (2) w kodzie IA-64 p3 jest prawdziwy tylko wówczas, gdy zewnętrzna instrukcja **if** w kodzie źródłowym jest prawdziwa. To znaczy, że predykat p3 jest prawdziwy tylko wówczas, gdy wyrażenie (a AND b) jest prawdziwe (np. a ≠ 0 AND b ≠ 0). Z tego powodu część **then** zewnętrznej instrukcji **if** podlega predykacji w p3. Rozkaz (4) w kodzie IA-64 decyduje, czy zostanie wykonany rozkaz odejmowania w zewnętrznej części **else**. Na zakończenie bezwarunkowo następuje inkrementacja i. Jeśli spojrzymy na kod źródłowy, następnie zaś na predykatywny to przekonamy się, że tylko jeden z rozkazów (3), (5) i (6) zostanie wykonany. W zwykłym procesorze superskalarnym do odgadnięcia właściwego rozkazu posłużylibyśmy się przewidywaniem rozgałęzień, po czym podążalibyśmy odpowiadającą mu ścieżką. Gdyby procesor odgadł niewłaściwie, potok musiałby być opróżniony. Procesor IA-64 może rozpocząć wykonywanie wszystkich trzech rozkazów, po czym – gdy staną się znane wartości rejestrów predykacji – odesłać jedynie wyniki ważnego rozkazu. Korzystamy więc z dodatkowych, równoległych jednostek wykonujących, aby uniknąć opóźnień towarzyszących opróżnianiu potoku.

Wiele początkowych badań na temat wykonywania predykatywnego realizowano na University of Illinois. Przeprowadzone tam badania symulacyjne wykazały, że użycie predykacji prowadzi do znacznego ograniczenia rozgałęzień dynamicznych i niewłaściwych przewidywań rozgałęzień, a także do znacznego zwiększenia wydajności w przypadku procesorów o wielu równoległych potokach (np. [MAHL94], [MAHL95]).

Spekulacja sterowaniem

Kolejną kluczową innowacją w IA-64 jest spekulacja sterowaniem, znana również jako *ładowanie spekulatywne*. Umożliwia ono procesorowi ładowanie danych z pamięci, zanim będzie ich potrzebował program, co ma na celu uniknięcie opóźnień związanych z czasem oczekiwania na wynik ładowania. Procesor opóźnia również informowanie o wyjątkach aż do chwili, w której staje się to konieczne. Przenoszenie rozkazu ładowania do wcześniejszego miejsca w strumieniu rozkazów jest określane jako *przerzucanie* (*hoist*).

Minimalizacja opóźnień ładowania ma kluczowe znaczenie dla podniesienia wydajności. Zwykle na początku bloku kodu znajduje się wiele operacji ładowania polegających na przenoszeniu danych z pamięci do rejestrów. Ponieważ pamięć – nawet wyposażona w dwupoziomową pamięć podręczną – jest powolna w porównaniu z procesorem, opóźnienia w uzyskiwaniu danych z pamięci mogą się stać wąskim gardłem. Aby to zminimalizować, dobrze byłoby tak przeorganizować kod, żeby operacje ładowania były realizowane jak najwcześniej. Do pewnego stopnia można tego dokonać za pomocą dowolnego kompilatora. Problemy pojawiają się wówczas, gdy próbujemy przenieść ładowanie poprzez strumień sterowania. Nie można bezwarunkowo przenieść ładowania ponad rozgałęzieniem, ponieważ w rzeczywistości ładowanie może nie nastąpić. Moglibyśmy przenieść ładowanie warunkowo, posługując się predykatami, dzięki czemu dane mogłyby być pobierane z pamięci, jednak nie byłyby kierowane do rejestru architektonicznego, dopóki nie byłby znany wynik predykacji; moglibyśmy też użyć metody przewidywania rozgałęzień, którą poznaliśmy w rozdz. 14. Problem towarzyszący takim rozwiązaniom polega jednak na tym, że ładowanie mogłoby spowodować niepożądane skutki. Mógłby być wygenerowany wyjątek związany z niewłaściwym adresem lub błędem strony. Jeśli tak by się stało, procesor musiałby sobie poradzić z wyjątkiem lub błędem, co wywołałoby opóźnienie.

W jaki sposób moglibyśmy więc przenieść ładowanie ponad rozgałęzieniem? Rozwiązaniem przyjętym w IA-64 jest spekulacja sterowaniem, która pozwala oddzielić samo ładowanie (dostarczenie wartości) od wyjątku (rys. 15.3b). Rozkaz ładowania w programie oryginalnym jest zastępowany przez dwa rozkazy:

- Rozkaz spekulatywnego ładowania (ld.s) obejmuje pobranie z pamięci, wykrywanie wyjątku, lecz nie obejmuje dostarczenia wyjątku (wywołania procedury przetwarzającej wyjątek). Rozkaz ten jest przerzucany do odpowiedniego, wcześniejszego punktu w programie.
- Rozkaz sprawdzenia (chk.s) jest pozostawiany w miejscu oryginalnego ładowania i dostarcza wyjątki. Może on podlegać predykacji, dzięki czemu zostanie wykonany tylko wówczas, gdy predykat jest prawdziwy.

Jeśli ld.s wykryje wyjątek, ustawia bit znacznika (*token*) związany z rejestrem docelowym, znany jako bit NaT (*Not a Thing*, coś nieodpowiedniego). Jeśli zostanie wykonany odpowiedni rozkaz chk.s i jeśli bit NaT będzie ustawiony, nastąpi rozgałęzienie do procedury przetwarzania wyjątków.

Zapoznajmy się z prostym przykładem zaczerpniętym z [INTE00a, vol. 1]. Oto program oryginalny:

```
(p1)  br some_label              // Cykl 0
      ld8 r1 = [r5] ;;           // Cykl 1
      add r2 = r1, r3            // Cykl 3
```

Pierwszy rozkaz prowadzi do rozgałęzienia, jeśli predykat p1 jest prawdziwy (rejestr p1 ma wartość 1). Zauważmy, że rozkazy rozgałęzienia i ładowania znajdują się w tej samej grupie, mimo że ładowanie nie może być wykonane, zanim nie nastąpi rozgałęzienie. IA-64 gwarantuje, że jeśli rozgałęzienie nastąpi, to późniejsze rozkazy – nawet znajdujące się w tej samej grupie – nie zostaną wykonane. W celu zwiększenia skuteczności w implementacjach IA-64 może być użyte przewidywanie rozgałęzień, jednak muszą być podjęte środki zapobiegające niewłaściwym wynikom. Zauważmy wreszcie, że rozkaz dodawania (add) zostaje opóźniony co najmniej o jeden okres zegara (jeden cykl) ze względu na opóźnienie w pamięci towarzyszące operacji ładowania.

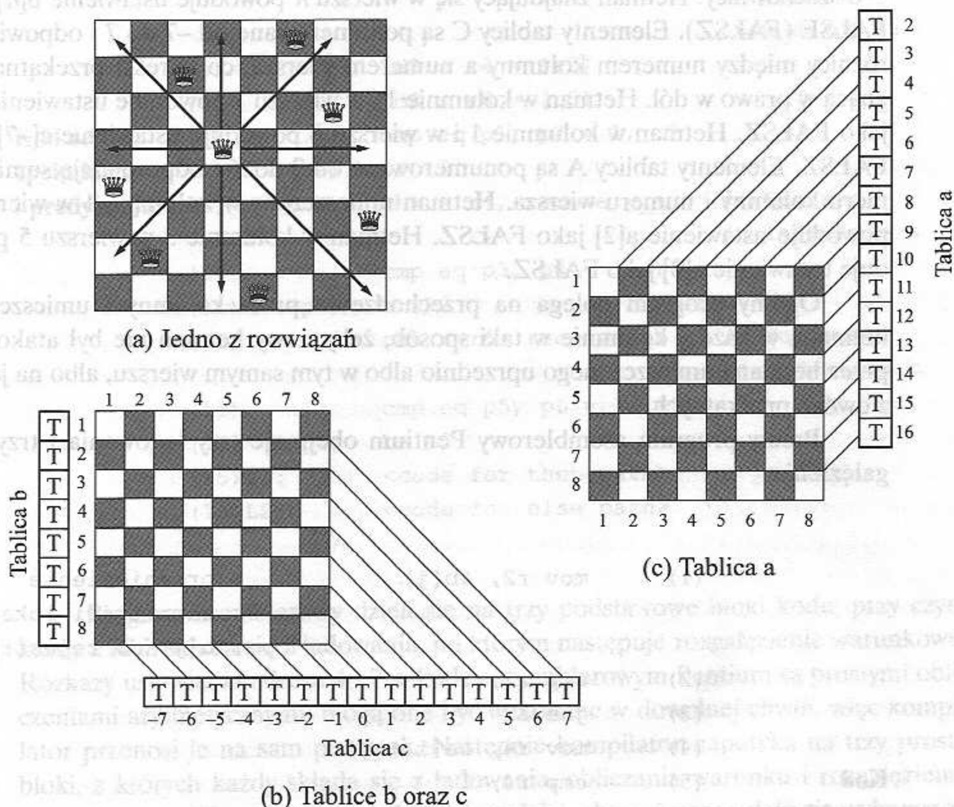
Kompilator może zmienić ten program, posługując się spekulatywnym ładowaniem i sprawdzaniem:

```
      ld8.s r1 = [r5] ;;         // Cykl -2
                                 // Inne rozkazy
(p1)  br some_label              // Cykl 0
      chk.s r1, recovery         // Cykl 0
      add r2 = r1, r3            // Cykl 0
```

Nie możemy po prostu przenieść rozkazu ładowania ponad rozkazem rozgałęzienia, ponieważ rozkaz ładowania może spowodować wyjątek (np. r5 może zawierać wskaźnik zerowy). Zamiast tego przekształcamy ładowanie na ładowanie spekulatywne – `ld8.s` – i dopiero wówczas je przenosimy. Ładowanie spekulatywne nie sygnalizuje natychmiastowo wyjątku tuż po jego wykryciu; rejestruje po prostu ten fakt, ustawiając bit NaT w rejestrze docelowym (w tym przypadku r1). W ramach ładowania spekulatywnego są wykonywane bezwarunkowo przynajmniej dwa cykle przed rozgałęzieniem. Następnie rozkaz `chk.s` sprawdza, czy bit NaT w r1 został ustawiony jako 1. Jeśli nie, jest wykonywany kolejny rozkaz. Jeśli tak, następuje rozgałęzienie do programu przywracania. Zauważmy, że wszystkie rozkazy rozgałęzienia, sprawdzenia i dodawania są pokazane jako wykonywane w tym samym cyklu. Rozwiązania sprzętowe zapewniają jednak, że wyniki ładowania spekulatywnego nie aktualizują stanu aplikacji (nie zmieniają zawartości r1 i r2), dopóki nie wystąpią dwa warunki: rozgałęzienie nie następuje (p1 = 0), a sprawdzenie nie wykrywa wstrzymanego wyjątku (r1.NaT = 0).

Trzeba zwrócić uwagę na jeszcze jeden ważny element tego przykładu. Jeśli nie ma wyjątku, to ładowanie spekulatywne jest ładowaniem rzeczywistym i jest realizowane przed rozgałęzieniem, które może wystąpić. Jeśli to rozgałęzienie zacho-

dzi, to – jak się okazuje – zostało zrealizowane ładowanie nieprzewidziane w oryginalnym programie. W napisanym programie zakładało się, że na ścieżce dokonanego rozgałęzienia rejestr r1 nie jest odczytywany. Jeśli w rzeczywistości zawartość tego rejestru jest odczytywana mimo wykonania rozgałęzienia, to kompilator musi użyć innego rejestru do zapisania wyniku spekulatywnego.



Rysunek 15.5. Problem ośmiu hetmanów

Zapoznajmy się teraz z bardziej złożonym przykładem, używanym przez Intela i HP do testowania wzorcowego programów predykatywnych oraz do demonstrowania ładowania spekulatywnego, znanym jako *problem ośmiu hetmanów*. Celem jest takie ustawienie ośmiu hetmanów na szachownicy, aby żadna z nich nie zagrażała żadnej z pozostałych. Na rysunku 15.5a pokazano jedno z rozwiązań. Kluczowy wiersz kodu źródłowego w pętli wewnętrznej jest następujący:

```
if ((b[j] == true) && (a[i+j] == true) && (c[i-j] == true))
```

gdzie $1 \leqslant i, j \leqslant 8$.

Mechanizm wykrywania konfliktu hetmanów składa się z 3 tablic boolowskich, które służą do wykrywania statusu hetmanów w każdym wierszu i każdej przekątnej. TRUE oznacza, że w danym wierszu lub przekątnej nie ma żadnego hetmana; FALSE oznacza, że hetman już tu jest. Na rysunkach 15.5(b) i (c) pokazano odwzorowanie tych tablic na szachownicy. Wszystkie elementy tablicy są inicjowane jako TRUE (PRAWDA). Elementy od 1 do 8 tablicy B odpowiadają wierszom 1–8 szachownicy. Hetman znajdujący się w wierszu *n* powoduje ustawienie b[*n*] jako FALSE (FAŁSZ). Elementy tablicy C są ponumerowane od –7 do 7 i odpowiadają różnicy między numerem kolumny a numerem wiersza, co określa przekątną biegnącą w prawo w dół. Hetman w kolumnie 1 i w wierszu 1 powoduje ustawienie c[0] jako FAŁSZ. Hetman w kolumnie 1 i w wierszu 8 powoduje ustawienie c[–7] jako FAŁSZ. Elementy tablicy A są ponumerowane od 2 do16 i odpowiadają sumie numeru kolumny i numeru wiersza. Hetman umieszczony w kolumnie 1 w wierszu 1 powoduje ustawienie a[2] jako FAŁSZ. Hetman w kolumnie 3 w wierszu 5 powoduje ustawienie a[8] jako FAŁSZ.

Ogólny program polega na przechodzeniu przez kolumny i umieszczaniu hetmana w każdej kolumnie w taki sposób, żeby nowy hetman nie był atakowany przez hetmana umieszczonego uprzednio albo w tym samym wierszu, albo na jednej z dwóch przekątnych.

Prosty program asemblerowy Pentium obejmuje trzy ładowania i trzy rozgałęzienia:

Kod asemblerowy:

```
(1)      mov r2, &b[j]         ; przeniesienie
                               ; zawartości lokacji
                               ; b[j] do rejestru r2
(2)      cmp r2, 1
(3)      jne L2
(4)      mov r4, &a[i + j]
(5)      cmp r4, 1
(6)      jne L2
(7)      mov r6, &c[i - j]
(8)      cmp r6, 1
(9)      jne L2
(10)L1: <code for then path>
(11)L2: <code for else path>
```

W powyższym programie notacja &x symbolizuje adres natychmiastowy dla lokacji x. Zastosowanie spekulatywnego ładowania i wykonywania predykatywnego prowadzi do następującego programu:

Kod ze spekulacją i predykacją:

```
(1)           mov r1 = &b[j]            // Przeniesienie
                                        // adresu b[j]
                                        // do r1
(2)           mov r3 = &a[i + j]
(3)           mov r5 = &c[i - j + 7]
(4)           ld8 r2 = [r1]             // Ładowanie
                                        // pośrednie
                                        // poprzez r1
(5)           ld8.s r4 = [r3]
(6)           ld8.s r6 = [r5]
(7)           cmp.eq p1, p2 = 1, r2
(8)     (p2)  br L2
(9)           chk.s r4, recovery_a      // Naprawa
                                        // ładowania a
(10)          cmp.eq p3, p4 = 1, r4
(11)    (p4)  br L2
(12)          chk.s r6, recovery_b      // Naprawa
                                        //ładowania b
(13)          cmp.eq p5, p6 = 1, r5
(14)    (p6)  br L2
(15)L1:       <code for then path>
(16)L2:       <code for else path>
```

Program asemblerowy dzieli się na trzy podstawowe bloki kodu, przy czym każdy z nich składa się z ładowania, po którym następuje rozgałęzienie warunkowe. Rozkazy ustawiania adresu 4 i 7 w kodzie asemblerowym Pentium są prostymi obliczeniami arytmetycznymi; mogą one być wykonane w dowolnej chwili, więc kompilator przenosi je na sam początek. Następnie kompilator napotyka na trzy proste bloki, z których każdy składa się z ładowania, obliczania warunku i rozgałęzienia warunkowego. Nie można mieć zbytnich nadziei, aby coś z tego dało się wykonywać równolegle. Ponadto, jeśli założymy, że ładowanie zajmuje dwa lub więcej cykli zegara, stracimy pewien czas, zanim jeszcze zostanie wykonane rozgałęzienie warunkowe. Czego natomiast może dokonać kompilator, to przerzucenia drugiego i trzeciego ładowania (rozkazy 5 i 8 w kodzie Pentium) ponad wszystkimi rozgałęzieniami. Realizuje się to, umieszczając ładowanie spekulatywne bliżej początku (rozkazy 5 i 6 IA-64) i pozostawienie rozkazu sprawdzenia w ich dotychczasowym miejscu (rozkazy 9 i 12 IA-64).

Transformacja taka umożliwia wykonywanie wszystkich trzech ładowań równolegle i na tyle wczesne ich rozpoczynanie, aby zminimalizować opóźnienia lub uniknąć ich w ogóle. Kompilator może się posunąć dalej agresywnie, posługując się predykacją, i wyeliminować dzięki temu dwa spośród trzech rozgałęzień:

Zrewidowany kod ze spekulacją i predykacją:

```
(1)               mov r1 = &b[j]
(2)               mov r3 = &a[i + j]
(3)               mov r5 = &c[i - j + 7]
(4)               ld8 r2 = [r1]
(5)               ld8.s r4 = [r3]
(6)               ld8.s r6 = [r5]
(7)               cmp.eq p1, p2 = 1, r2
(8)       (p1)    chk.s r4, recovery_a
(9)       (p1)    cmp.eq p3, p4 = 1, r4
(10)      (p3)    chk.s r6, recovery_b
(11)      (p3)    cmp.eq p5, p4 = 1, r5
(12)      (p6)    br L2
(13)L1:           <code for then path>
(14)L2:           <code for else path>
```

Mieliśmy już rozkaz porównania, który generował dwa predykaty. W kodzie zrewidowanym, zamiast rozgałęziania na podstawie fałszywego predykatu, kompilator kwalifikuje wykonywanie obydwu sprawdzeń, po czym następuje porównanie na podstawie predykatu prawdziwego. Wyeliminowanie dwóch rozgałęzień oznacza wyeliminowanie dwóch potencjalnie błędnych przewidywań, oszczędności są więc większe niż tylko wyeliminowanie dwóch rozkazów.

Spekulacja danymi

W przypadku spekulacji sterowaniem ładowanie jest przenoszone bliżej początku sekwencji kodu w celu skompensowania opóźnienia towarzyszącego ładowaniu, po czym – jeśli następnie okazuje się, że ładowanie nie miało miejsca – dokonuje się sprawdzenia dla upewnienia się, czy nie wystąpił wyjątek. W przypadku spekulacji danymi ładowanie jest przenoszone przed rozkaz zapisu, który mógłby zmienić lokację pamięci będącą źródłem tego ładowania. Następnie dokonuje się sprawdzenia w celu upewnienia się, że ładowanie odbiera właściwą wartość z pamięci. W celu objaśnienia tego mechanizmu posłużymy się przykładem zaczerpniętym z [INTE00a, vol. 1]:

Rozważmy następujący fragment programu:

```
st8 [r4] = r12              // Cykl 0
ld8 r6 = [r8] ;;            // Cykl 0
add r5 = r6, r7 ;;          // Cykl 2
st8 [r18] = r5              // Cykl 3
```

W takim stanie kod wymaga do wykonania czterech cykli rozkazu. Jeśli rejestry r4 i r8 nie zawierają takiego samego adresu pamięci, to zapisywanie poprzez r4 nie może wpłynąć na wartość adresu zawartego w r8; w tych warunkach bezpieczniej jest zmienić kolejność ładowania i zapisu, aby szybciej doprowadzić do r6 wartość, która następnie będzie potrzebna. Ponieważ jednak adresy w r4 i r8 mogą być takie same

lub mogą się nakładać, taka zamiana nie jest bezpieczna. W IA-64 pokonuje się tę trudność za pomocą metody zwanej *ładowaniem wyprzedzającym* (*advanced load*).

```
ld8.a r6 = [r8] ;;          // Cykl -2 lub wcześniejszy;
                            // ładowanie wyprzedzające
                            // pozostałych rozkazów
st8 [r4] = r12              // Cykl 0
ld8.c r6 = [r8]             // Cykl 0; sprawdzenie ładowania
add r5 = r6, r7 ;;          // Cykl 0
st8 [r18] = r5              // Cykl 1
```

Przenieśliśmy tutaj rozkaz `ld` bliżej początku i przekształciliśmy go na ładowanie wyprzedzające. Poza wykonaniem określonego ładowania, rozkaz `ld8.a` zapisuje jego adres źródłowy (zawarty w r8) w sprzętowej strukturze danych znanej jako *tabela adresów ładowania wyprzedzającego* (*advanced load address table* – ALAT). Każdy rozkaz zapisu IA-64 sprawdza ALAT w poszukiwaniu wpisów, które nakładają się z jego adresem docelowym; jeśli stwierdza się zgodność, wpis ALAT jest usuwany. Gdy oryginalny rozkaz `ld8` zostaje przekształcony na `ld8.a` i przeniesiony, w początkowym miejscu tego rozkazu jest umieszczany rozkaz sprawdzenia ładowania, `ld8.c`. Podczas wykonywania tego ostatniego rozkazu sprawdza się ALAT w poszukiwaniu adresów zgodnych. Jeśli taki adres zostanie znaleziony, żaden rozkaz zapisu między ładowaniem wyprzedzającym a sprawdzeniem ładowania nie zmienił źródłowego adresu ładowania, i wówczas nie jest podejmowane żadne działanie. Jeśli jednak rozkaz sprawdzenia ładowania nie doprowadzi do znalezienia zgodnego wpisu w ALAT, operacja ładowania jest przeprowadzana ponownie, aby zapewnić właściwy wynik.

Możemy również być zainteresowani spekulatywnym wykonywaniem rozkazów, które wykazują zależność danych w stosunku do rozkazu ładowania, łącznie z samym ładowaniem. Wychodząc z tego samego programu początkowego, załóżmy, że przenosimy zarówno rozkaz ładowania, jak i następujący po nim rozkaz dodawania:

```
          ld8.a r6 = [r8] ;;    // Cykl -3 lub wcześniejszy;
                                // wyprzedzające ładowanie innych
                                // rozkazów
          add r5 = r6, r7       // Cykl -1; dodawanie z użyciem r6
                                // Inne rozkazy
          st8 [r4] = r12        // Cykl 0
          chk.a r6, recover     // Cykl 0; sprawdzenie
back:                           // Punkt powrotu ze skoku do
                                // przywracania
          st8 [r18] = r5        // Cykl 0
```

Zastosowaliśmy tutaj rozkaz `chk.a` zamiast rozkazu `ld8.c` w celu uprawomocnienia ładowania wyprzedzającego. Jeśli w wyniku wykonania rozkazu `chk.a` zostanie

stwierdzone, że ładowanie się nie powiodło, nie może ono po prostu być wykonane ponownie; zamiast tego następuje rozgałęzienie do procedury przywracania:

```
Recover:
        ld8 r6 = [r8] ;;          // ponowne załadowanie r6 z [r8]
        add r5 = r6, r7, ;;       // ponowne wykonanie dodawania
        br back                   // skok z powrotem do kodu głównego
```

Sposób ten jest skuteczny jedynie wówczas, gdy istnieje niewielkie prawdopodobieństwo nakładania się ładowań i zapisów.

Potokowanie programowe

Rozważmy następującą pętlę:

```
L1: ld4 r4 = [r5], 4 ;;        // Cykl 0; ładowanie postinc 4
    add r7 = r4, r9 ;;         // Cykl 2
    st4 [r6] = r7, 4           // Cykl 3; zapisywanie postinc 4
    br.cloop L1 ;;             // Cykl 3
```

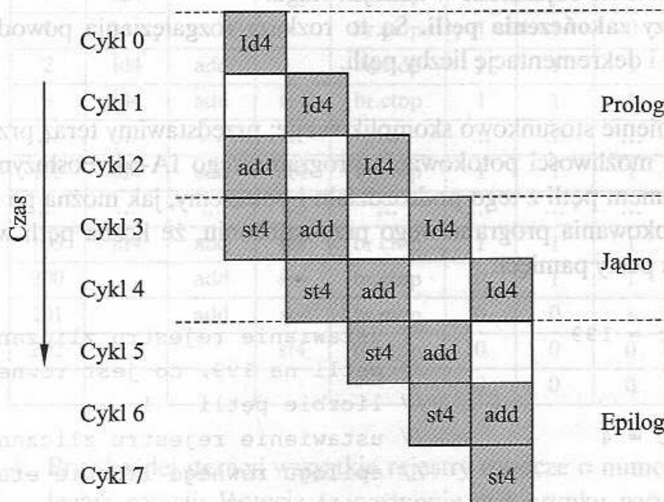
Pętla ta polega na dodaniu stałej do jednego wektora i zapisaniu wyniku w innym wektorze (np. y[i] = x[i] + c). Rozkaz `ld4` powoduje załadowanie 4 bajtów z pamięci. Kwalifikator „, 4” na końcu rozkazu sygnalizuje, że jest to forma podstawowej aktualizacji rozkazu ładowania; po dokonaniu ładowania adres w r5 jest zwiększany o 4. Podobnie rozkaz `st4` powoduje zapisanie 4 bajtów w pamięci, a po dokonaniu zapisu adres w r6 jest zwiększany o 4. Rozkaz `br.cloop` – znany jako *rozgałęzienie ze zliczaniem pętli* – używa licznika aplikacyjnego *liczby pętli* (*loop count* – LC). Jeśli wartość w rejestrze LC jest większa od zera, jest ona dekrementowana i zachodzi rozgałęzienie. Początkową wartością LC jest liczba iteracji pętli.

Zwróćmy uwagę, że w tym programie wewnątrz pętli praktycznie nie ma możliwości paralelizmu na poziomie rozkazu. Co więcej, wszystkie rozkazy w iteracji x są wykonywane przed rozpoczęciem iteracji x + 1. Jeśli jednak nie ma konfliktu adresów między ładowaniem a zapisem (r5 i r6 wskazują na nienakładające się lokacje pamięci), można byłoby poprawić wykorzystanie czasu, przenosząc niezależne rozkazy z iteracji x + 1 do iteracji x. Można to wyrazić inaczej: gdybyśmy zmodyfikowali kod pętli, pisząc w istocie nowy zbiór rozkazów dla każdej iteracji, to powstałyby warunki do zwiększenia paralelizmu. Spójrzmy, co można byłoby osiągnąć w przypadku pięciu iteracji:

```
ld4 r32 = [r5], 4 ;;           // Cykl 0
ld4 r33 = [r5], 4 ;;           // Cykl 1
ld4 r34 = [r5], 4              // Cykl 2
add r36 = r32, r9 ;;           // Cykl 2
ld4 r35 = [r5], 4              // Cykl 3
```

```
add r37 = r33, r9          // Cykl 3
st4 [r6] = r36, 4 ;;       // Cykl 3
ld4 r36 = [r5], 4          // Cykl 3
add r38 = r34, r9          // Cykl 4
st4 [r6] = r37, 4 ;;       // Cykl 4
add r39 = r35, r9          // Cykl 5
st4 [r6] = r38, 4 ;;       // Cykl 5
add r40 = r36, r9          // Cykl 6
st4 [r6] = r39, 4 ;;       // Cykl 6
st4 [r6] = r40, 4 ;;       // Cykl 7
```

Program ten polega na przeprowadzeniu 5 iteracji w 7 cyklach, w porównaniu z 20 cyklami w oryginalnym programie pętli. Zakłada się przy tym, że istnieją dwa porty pamięci, dzięki czemu jest możliwe równoległe wykonywanie ładowania i zapisu. Jest to przykład potokowania programowego, analogicznego do potokowania sprzętowego. Proces ten zilustrowano na rys. 15.6. Paralelizm osiąga się, grupując rozkazy pochodzące z różnych iteracji. Aby to funkcjonowało, rejestry tymczasowe, używane wewnątrz pętli, muszą być zmieniane dla każdej iteracji, aby zapobiec konfliktom rejestrowym. W tym przypadku zostały użyte dwa rejestry tymczasowe (r4 i r7 w programie oryginalnym). W programie rozszerzonym po każdej iteracji numer rejestru jest zwiększany, zaś numery rejestrów są inicjowane jako wystarczająco odległe, aby zapobiec nakładaniu się.



Rysunek 15.6. Przykład potokowania programowego

Na rysunku 15.6 pokazano, że w potoku programowym występują trzy fazy. Podczas **fazy prologu** jest inicjowana nowa iteracja, przy czym cykl zegara i potok stopniowo wypełniają się. Podczas **fazy jądra** potok jest pełny, co oznacza maksymalny paralelizm. W naszym przykładzie podczas fazy jądra są wykonywane równo-

legle trzy rozkazy, jednak szerokość potoku wynosi 4. Podczas **fazy epilogu** w każdym cyklu zegara jest kończona jedna iteracja.

Potokowanie programowe poprzez rozwijanie pętli nakłada na kompilator lub na programistę obowiązek właściwego przypisywania nazw rejestrów. Co więcej, w przypadku długich pętli o wielu iteracjach rozwijanie prowadzi do znacznego zwiększenia rozmiarów kodu. W przypadku pętli nieokreślonej (całkowita liczba iteracji nie jest znana podczas kompilowania) zadanie to jest jeszcze bardziej skomplikowane ze względu na konieczność częściowego rozwijania pętli, następnie zaś kontrolowania ich liczby. IA-64 zapewnia wsparcie sprzętowe potokowania programowego bez rozszerzania kodu i przy minimalnych dodatkowych obciążeniach dla kompilatora. Podstawowe właściwości wspierające potokowanie programowe to:

- **Automatyczne przemianowywanie rejestrów.** Obszar tablic rejestrów predykatywnych i zmiennopozycyjnych o ustalonym rozmiarze (p16 do p63; fr32 do fr127) oraz obszar tablicy rejestrów roboczych o rozmiarze programowanym (maksymalny zakres od r32 do r127) umożliwiają rotację. Oznacza to, że podczas każdej iteracji pętli w potoku programowym odniesienia do rejestrów w ramach tych zakresów są inkrementowane automatycznie. Jeśli zatem w pierwszej iteracji pętla wykorzystuje rejestr roboczy r32, w drugiej iteracji automatycznie używa r33 itd.
- **Predykacja.** Każdy rozkaz w pętli podlega predykacji w rotacyjnym rejestrze predykatywnym. Ma to na celu określenie, czy potok jest w fazie prologu, jądra, czy też epilogu, co zostanie objaśnione w dalszym ciągu.
- **Specjalne rozkazy zakończenia pętli.** Są to rozkazy rozgałęziania powodujące rotację rejestrów i dekrementację liczby pętli.

Jest to zagadnienie stosunkowo skomplikowane; przedstawimy teraz przykład ilustrujący niektóre możliwości potokowania programowego IA-64. Posłużymy się oryginalnym programem pętli z tego podrozdziału i pokażemy, jak można go zmienić pod kątem potokowania programowego przy założeniu, że liczba pętli wynosi 200 i że istnieją dwa porty pamięci:

```
          mov lc = 199             // ustawianie rejestru zliczania
                                   // pętli na 199, co jest równe
                                   // liczbie pętli – 1
          mov ec = 4               // ustawienie rejestru zliczania
                                   // epilogu równego liczbie etapów
                                   // epilogu + 1
          mov pr.rot = 1<<16;;     // pr16 = 1; reszta = 0
L1: (p16) ld4 r32 = [r5], 4        // Cykl = 0
    (p17) ---                      // Etap pusty
    (p18) add r35 = r34, r9        // Cykl 0
    (p19) st4 [r6] = r36, 4        // Cykl 0
          br.ctop L1 ;;            // Cykl 0
```

Przedstawimy teraz kluczowe kwestie związane z tym programem:

1. Program pętli jest dzielony na wiele *etapów*, przy czym na jeden etap przypada od zera do wielu rozkazów.
2. Wykonywanie pętli przechodzi przez trzy fazy. Podczas fazy prologu w każdym cyklu jest uruchamiana nowa iteracja, co powoduje dodanie jednego etapu do potoku. Podczas fazy jądra w każdym cyklu jedna iteracja pętlowa jest rozpoczynana i jedna jest kończona; potok jest wypełniony i maksymalna liczba etapów jest aktywna. Podczas fazy epilogu nie są rozpoczynane żadne nowe iteracje i za każdym cyklem jedna iteracja jest kończona, co powoduje opróżnianie potoku programowego.
3. Każdemu etapowi jest przypisywany predykat w celu sterowania aktywowaniem rozkazów na tym etapie. Podczas fazy prologu przy pierwszej iteracji p16 jest prawdziwy, a p17, p18 i p19 – fałszywe. Przy drugiej iteracji prawdziwe są p16 i p17, przy trzeciej zaś – p16, p17 i p18. Podczas fazy jądra wszystkie predykaty są prawdziwe. Podczas fazy epilogu predykaty kolejno są zmieniane na fałszywe, poczynając od p16. Zmiany wartości predykatów uzyskuje się dzięki rotacji rejestrów predykacji.

Tabela 15.4. Przebieg pętli w przykładzie potokowania programowego

Cykl	Jednostka wykonująca/Rozkaz				Stan przed br.ctop					
	M	I	M	B	P16	P17	P18	P19	LC	EC
0	ld4			br.ctop	1	0	0	0	199	4
1	ld4			br.ctop	1	1	0	0	198	4
2	ld4	add		br.ctop	1	1	1	0	197	4
3	ld4	add	st4	br.ctop	1	1	1	1	196	4
...	...	...	...	...	...	...	...	...	...	...
100	ld4	add	st4	br.ctop	1	1	1	1	99	4
...	...	...	...	...	...	...	...	...	...	...
199	ld4	add	st4	br.ctop	1	1	1	1	0	4
200		add	st4	br.ctop	0	1	1	1	0	3
201		add	st4	br.ctop	0	0	1	1	0	2
202			st4	br.ctop	0	0	0	1	0	1
					0	0	0	0	0	0

4. Przy każdej iteracji wszystkie rejestry robocze o numerach wyższych od 31 podlegają rotacji. Rotacja ta następuje w kierunku wyższych numerów rejestrów w sposób cykliczny. Na przykład wartość rejestru x po jednej rotacji zostanie umieszczona w rejestrze x+1; osiąga się to nie przez przenoszenie wartości, lecz przez sprzętowe przemianowanie rejestrów. Zatem w naszym przykładzie wartość, która podczas ładowania została zapisana w r32, po dwóch iteracjach (i dwóch rotacjach) jest odczytywana jako r34. Podobnie, wartość zapisana podczas dodawania w r35 po jednej iteracji jest odczytywana jako r36.

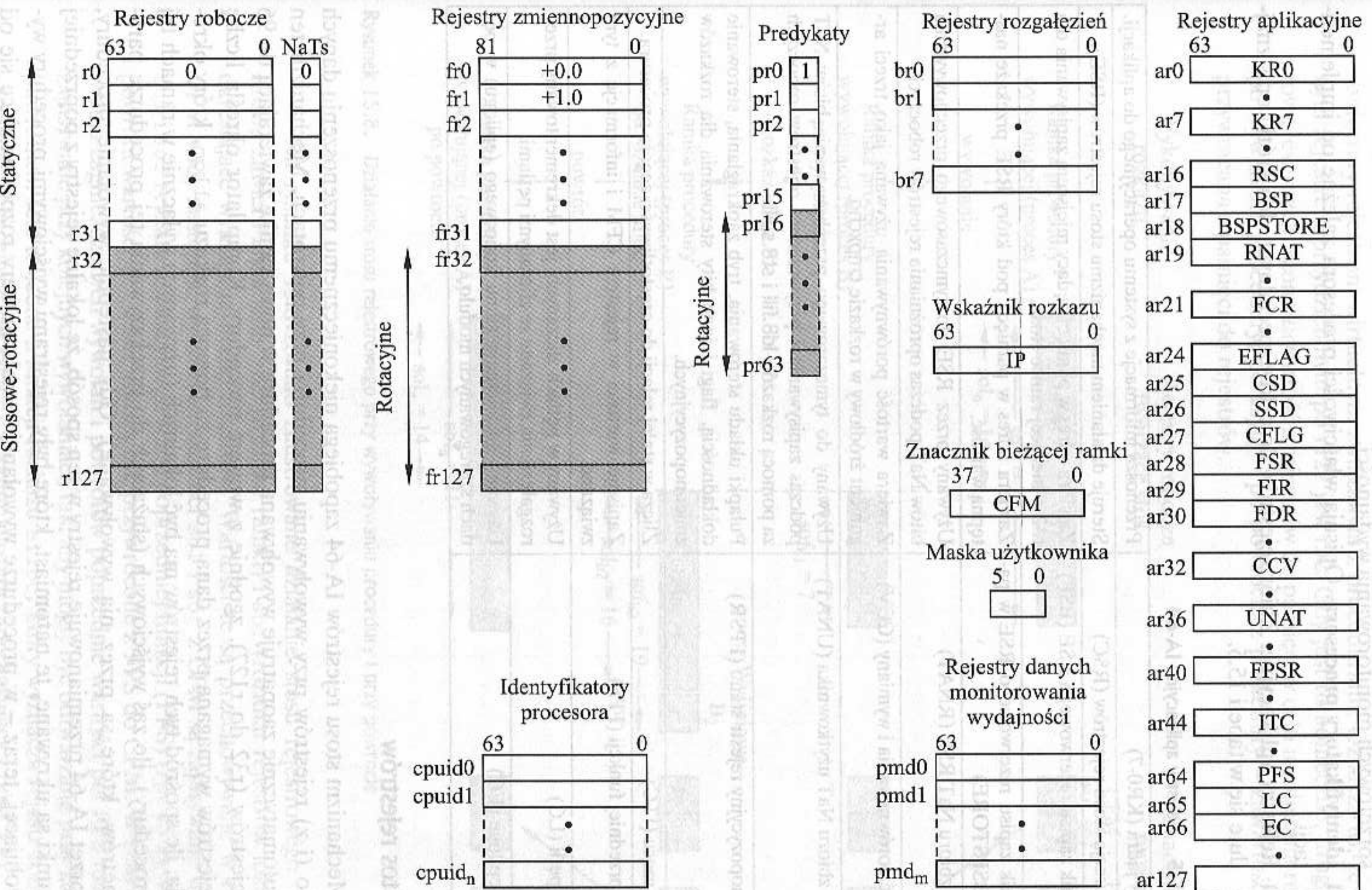
5. W przypadku rozkazu `br.ctop` rozgałęzienie następuje, gdy albo LC > 0, albo EC > 1. Wykonanie `br.ctop` wnosi następujące efekty dodatkowe: jeśli LC > 0, to LC ulega dekrementacji; dzieje się tak w fazach prologu i jądra. Jeśli LC = 0 i EC > 1, dekrementowany jest EC; dzieje się to w fazie epilogu. Rozkaz ten steruje również rotacją rejestrów. Jeśli LC > 0, każde wykonanie `br.ctop` umieszcza 1 w p63. W wyniku rotacji p63 staje się p16, wprowadzając ciągłą sekwencję jedynek do rejestrów predykacji w fazach prologu i jądra. Jeśli LC = 0, to `br.ctop` ustawia p63 na 0, wprowadzając zera do rejestrów predykacji podczas fazy epilogu.

Przebieg wykonywania rozkazów w tym przykładzie został pokazany w tabeli 15.4.

15.4. Architektura listy rozkazów IA-64

Na rysunku 15.7 został pokazany zbiór rejestrów dostępnych dla programów aplikacyjnych. Oznacza to, że rejestry te są widzialne dla aplikacji, mogą być odczytywane i w większości przypadków zapisywane. Zbiór rejestrów obejmuje:

- **Rejestry robocze.** 128 64-bitowych rejestrów roboczych. Z każdym rejestrem jest związany bit NaT służący do śledzenia wyjątków spekulatywnych, co zostało objaśnione w podrozdz. 15.3. Rejestry od r0 do r31 są określane jako statyczne; odniesienie programowe do któregokolwiek z nich jest interpretowane dosłownie. Rejestry od r32 do r127 mogą być używane jako rejestry rotacyjne dla potrzeb potokowania programowego (co zostanie jeszcze omówione w tym podrozdziale). Odniesienia do tych rejestrów są wirtualne. Rejestry te mogą być dynamicznie przemianowywane przez sprzęt.
- **Rejestry zmiennopozycyjne.** 128 82-bitowych rejestrów do zapisywania liczb zmiennopozycyjnych. Rozmiar ten wystarcza do zapisywania liczb w formacie podwójnie poszerzonym według IEEE 754 (patrz tabela 9.3). Rejestry od fr0 do fr31 są statyczne, rejestry zaś od fr32 do fr127 mogą być używane jako rotacyjne dla potrzeb potokowania programowego.
- **Rejestry predykacji.** 64 rejestry 1-bitowe służą jako predykaty. Rejestr pr0 jest zawsze ustawiany jako 1, aby umożliwić wykonywanie rozkazów nie objętych predykacją. Rejestry od pr0 do pr15 są statyczne, a rejestry od pr16 do pr63 mogą służyć jako rejestry rotacyjne dla potrzeb potokowania programowego.
- **Rejestry rozgałęzień.** 8 64-bitowych rejestrów używanych przy rozgałęzieniach.
- **Wskaźnik rozkazu.** Zawiera adres wiązki bieżąco wykonywanego rozkazu IA-64.
- **Znacznik bieżącej ramki.** Zawiera informację o stanie odnoszącą się do bieżącej ramki stosu rejestrów roboczych oraz informacje o rotacji dotyczące rejestrów fr i pr.
- **Maska użytkownika.** Zbiór wartości jednobitowych używany dla potrzeb wyrównywania pułapek, monitorowania wydajności i monitorowania wykorzystania rejestrów zmiennopozycyjnych.



Rysunek 15.7. Zbiór rejestrów aplikacyjnych IA-64

- **Rejestry danych monitorowania wydajności.** Używane do obsługiwania układów sprzętowych monitorowania wydajności.
- **Identyfikatory procesora.** Opisują właściwości procesora zależne od implementacji.
- **Rejestry aplikacyjne.** Zbiór rejestrów do celów specjalnych. Ich krótki opis znajduje się w tabeli 15.5.

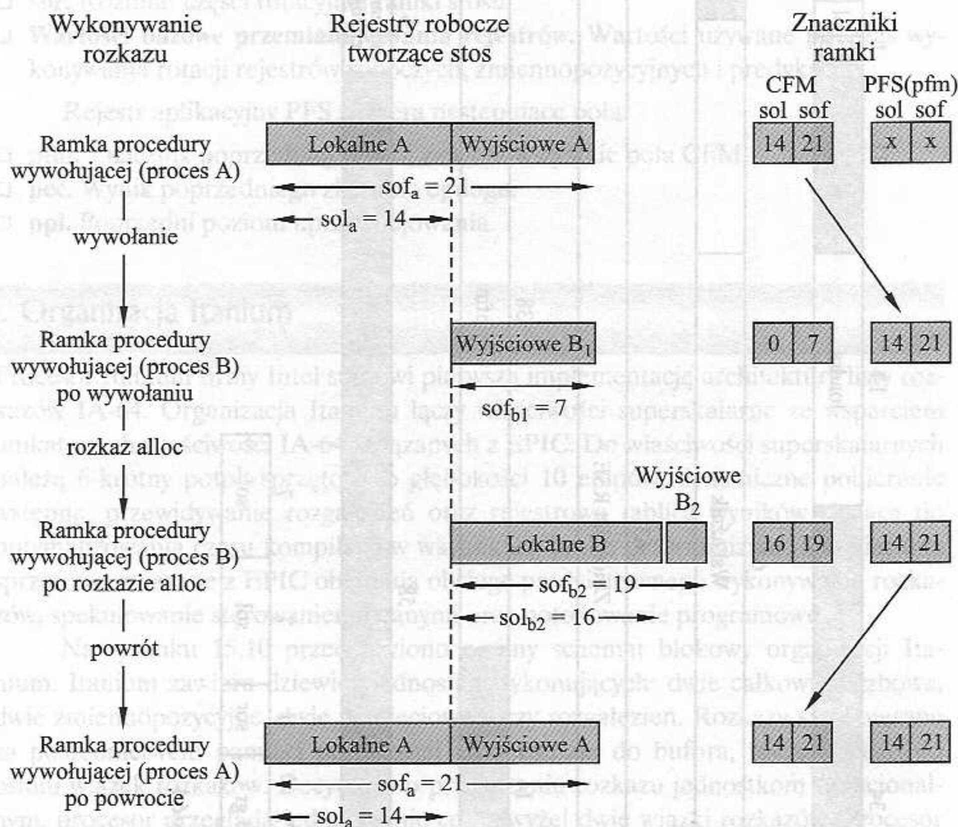
Tabela 15.5. Rejestry aplikacyjne IA-64

Rejestry jądra (KR0-7)	Przenoszą informacje z systemu operacyjnego do aplikacji.
Konfiguracja stosu rejestrów (RSC)	Steruje działaniem mechanizmu stosu rejestrów (RSE).
Wskaźnik zapisu rezerwowego RSE (BSP)	Zawiera adres w pamięci będący miejscem zapisywania dla r32 w bieżącej ramce stosu.
Wskaźnik zapisu rezerwowego RSE w pamięci (BSPSTORE)	Zawiera adres w pamięci, pod który RSE przekaże następną wartość.
Rejestr zbioru NaT RSE (RNAT)	Używany przez RSE do tymczasowego przechowywania bitów NaT podczas opróżniania rejestrów roboczych.
Wartość porównywania i wymiany (CCV)	Zawiera wartość porównywania używaną jako trzeci argument źródłowy w rozkazie cmpxchg.
Rejestr zbioru NaT użytkownika (UNAT)	Używany do tymczasowego przechowywania bitów NaT podczas zapisywania i przywracania rejestrów roboczych za pomocą rozkazów ld8.fill i st8.spill.
Zmiennopozycyjny rejestr stanu (FPSR)	Pułapki układu sterowania, tryb zaokrąglania, sterowanie dokładnością, flagi i inne bity sterowania dla rozkazów zmiennopozycyjnych.
Licznik czasu (ITC)	Zlicza w stałej relacji do częstotliwości zegara procesora.
Stan poprzedniej funkcji (FPS)	Zapisuje wartości w rejestrze CFM i informacje z tym związane.
Licznik pętli (LC)	Używany w pętlach zliczanych; jest dekrementowany przez rozgałęzienia związane ze zliczanymi pętlami.
Licznik epilogu (EC)	Używany do zliczania stanu końcowego (epilogu) w pętlach szeregowanych modulo.

Stos rejestrów

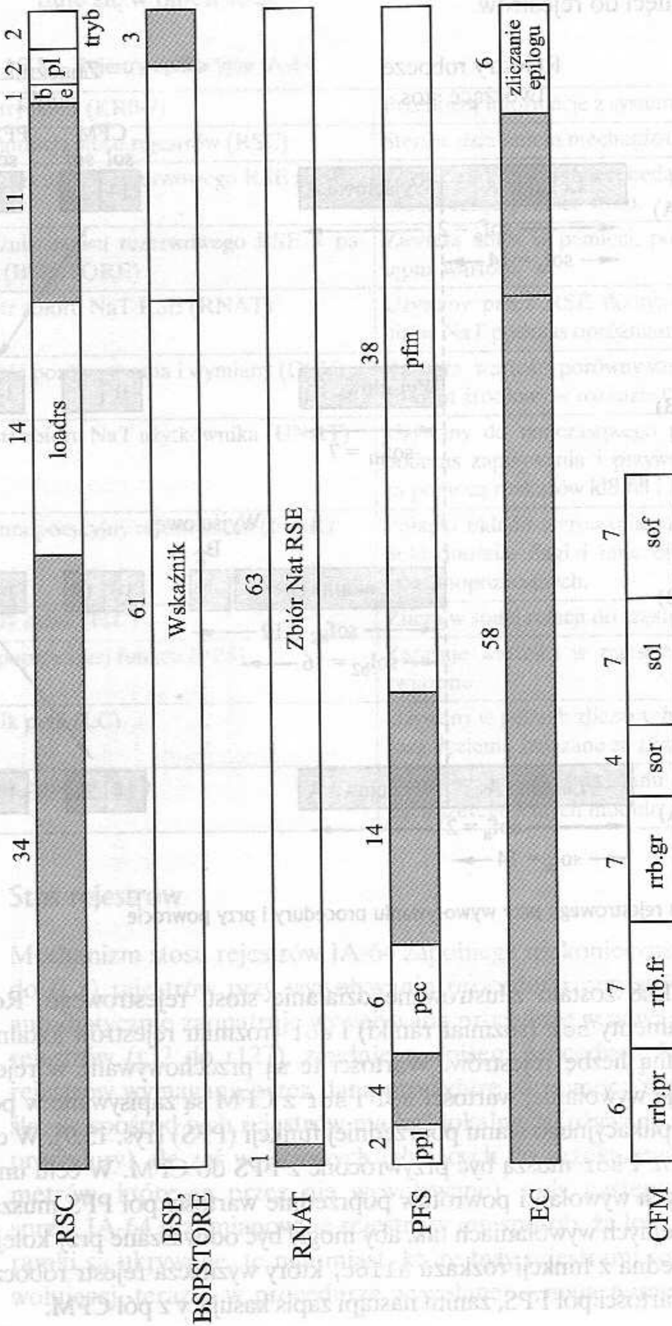
Mechanizm stosu rejestrów IA-64 zapobiega niekoniecznemu przenoszeniu danych do (i z) rejestrów przy wywoływaniu procedur i przy powrotach. Mechanizm ten automatycznie zaopatruje wywoływaną procedurę w nową **ramkę** zawierającą do 96 rejestrów (r32 do r127), zgodnie z wpisem procedury. Kompilator określa liczbę rejestrów wymaganą przez daną procedurę za pomocą rozkazu `alloc`, który określa, ile spośród tych rejestrów ma być lokalnych (używanych wyłącznie w ramach tej procedury), ile zaś wyjściowych (służących do przekazywania tej procedurze parametrów, które są przez nią wywoływane). Gdy następuje wywołanie procedury, sprzęt IA-64 przemianowuje rejestry w ten sposób, że lokalne rejestry z poprzedniej ramki są ukrywane, te natomiast, które były rejestrami wyjściowymi procedury wywołującej, teraz – w procedurze wywołanej – mają numery rozpoczynające się od

r32. Rejestry fizyczne w zakresie od r32 do r127 są przydzielane w sposób cykliczno-buforowy rejestrom wirtualnym związanym z procedurami. Oznacza to, że następnym rejestrem przydzielanym po r127 jest r32. Gdy jest to konieczne, drogą sprzętową jest przenoszona zawartość z rejestrów do pamięci w celu uwolnienia rejestrów po wywołaniu procedury, po czym – po powrocie z procedury – zawartość ta jest przywracana z pamięci do rejestrów.



Rysunek 15.8. Działanie stosu rejestrowego przy wywoływaniu procedury i przy powrocie

Na rysunku 15.8 zostało zilustrowane działanie stosu rejestrowego. Rozkaz `alloc` zawiera argumenty `sof` (rozmiar ramki) i `sol` (rozmiar rejestrów lokalnych), określające wymaganą liczbę rejestrów. Wartości te są przechowywane w rejestrze CFM. Gdy następuje wywołanie, wartości `sol` i `sof` z CFM są zapisywane w polach `sol` i `sof` rejestru aplikacyjnego stanu poprzedniej funkcji (PFS) (rys. 15.9). W czasie powrotu wartości `sol` i `sof` muszą być przywrócone z FPS do CFM. W celu umożliwienia zagnieżdżonych wywołań i powrotów poprzednie wartości pól PFS muszą być zapisywane przy kolejnych wywołaniach tak, aby mogły być odtwarzane przy kolejnych powrotach. Jest to jedna z funkcji rozkazu `alloc`, który wyznacza rejestr roboczy do zapisania bieżącej wartości pól FPS, zanim nastąpi zapis kasujący z pól CFM.



Rysunek 15.9. Formaty niektórych rejestrów IA-64

Znacznik bieżącej ramki i stan poprzedniej funkcji

Rejestr CFM opisuje stan bieżącej ramki stosu rejestrów roboczych, związanej z bieżąco czynną procedurą. Obejmuje on następujące pola:

- **sof.** Rozmiar ramki stosu.
- **sol.** Rozmiar części rejestrów lokalnych ramki stosu.
- **sor.** Rozmiar części rotacyjnej ramki stosu.
- **Wartości bazowe przemianowywania rejestrów.** Wartości używane podczas wykonywania rotacji rejestrów roboczych, zmiennopozycyjnych i predykacji.

Rejestr aplikacyjny PFS zawiera następujące pola:

- **pfm.** Znacznik poprzedniej ramki; zawiera wszystkie pola CFM.
- **pec.** Wynik poprzedniego zliczania epilogu.
- **ppl.** Poprzedni poziom uprzywilejowania.

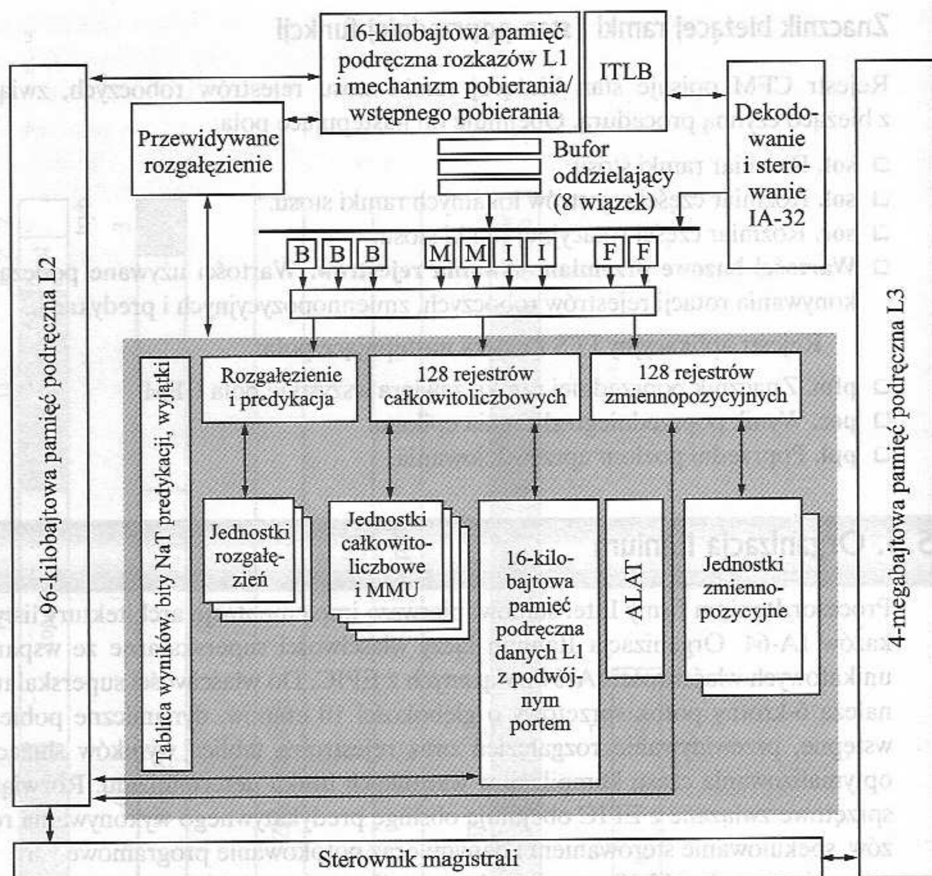
15.5. Organizacja Itanium

Procesor Itanium firmy Intel stanowi pierwszą implementację architektury listy rozkazów IA-64. Organizacja Itanium łączy właściwości superskalarne ze wsparciem unikatowych właściwości IA-64 związanych z EPIC. Do właściwości superskalarnych należą 6-krotny potok sprzętowy o głębokości 10 etapów, dynamiczne pobieranie wstępne, przewidywanie rozgałęzień oraz rejestrowa tablica wyników służąca do optymalizowania czasu kompilacji w warunkach braku determinizmu. Rozwiązania sprzętowe związane z EPIC obejmują obsługę predykatywnego wykonywania rozkazów, spekulowanie sterowaniem i danymi oraz potokowanie programowe.

Na rysunku 15.10 przedstawiono ogólny schemat blokowy organizacji Itanium. Itanium zawiera dziewięć jednostek wykonujących: dwie całkowitoliczbowe, dwie zmiennopozycyjne, dwie pamięciowe i trzy rozgałęzień. Rozkazy są pobierane za pośrednictwem pamięci podręcznej L1 i trafiają do bufora, mieszczącego do ośmiu wiązek rozkazów. Decydując o przekazaniu rozkazu jednostkom funkcjonalnym, procesor przegląda jednocześnie co najwyżej dwie wiązki rozkazów. Procesor może wydawać maksymalnie sześć rozkazów podczas cyklu zegara.

Organizacja ta jest pod pewnymi względami prostsza niż organizacja konwencjonalnych, współczesnych procesorów superskalarnych. W Itanium nie występują stacje rezerwacji, bufory zmiany kolejności ani bufory porządkowania pamięci; wszystkie te elementy zostały zastąpione prostszymi rozwiązaniami sprzętowymi dla potrzeb przetwarzania spekulatywnego. Rozwiązania sprzętowe odwzorowywania rejestrów są prostsze niż koordynowania nazw rejestrów, typowe dla maszyn superskalarnych. Nie ma układów logicznych wykrywania zależności rejestrowych, które są zastąpione przez jawne dyrektywy równoległego przetwarzania obliczone wstępnie przez oprogramowanie.

Posługując się przewidywaniem rozgałęzień, mechanizm pobierania i pobierania wstępnego może spekulatywnie ładować pamięć podręczną rozkazów L1 w celu zminimalizowania chybień w tej pamięci przy pobieraniu rozkazów. Pobrany kod jest kierowany do bufora oddzielającego, mieszczącego do ośmiu wiązek.



Rysunek 15.10. Organizacja procesora Itanium [SHAR00]

Zastosowano trzy poziomy pamięci podręcznej. Pamięć L1 jest podzielona na 16-kilobajtową pamięć rozkazów i 16-kilobajtową pamięć danych, z których każda jest 4-drożną pamięcią sekcyjno-skojarzeniową o rozmiarze wiersza 32 B. 96-kilobajtowa pamięć podręczna L2 jest 6-drożną pamięcią sekcyjno-skojarzeniową o wierszu 64 B. 4-megabajtowa pamięć podręczna L3 jest 4-drożną pamięcią sekcyjno-skojarzeniową o wierszu 64 B. Pamięci L1 i L2 znajdują się w mikroukładzie procesora; pamięć podręczna L3 jest poza tym mikroukładem, lecz w tej samej obudowie co procesor.

15.6. Polecana literatura i witryny WWW

[HUCK00] zawiera przegląd architektury IA-64; inny przegląd jest zawarty w [DULO98]. W [SCHL00a] znajduje się ogólne omówienie EPIC; bardziej wnikliwe ujęcie znajduje się w [SCHL00b]. Inne dobre ujęcia to [HWU01] i [KATH01]. [CHAS00] i [HWU98] zawierają

wprowadzenie do predykatywnego wykonywania rozkazów. Tom I [INTE00a] zawiera szczegółowy opis potokowania programowego; dwa artykuły zawierające dobre wyjaśnienie tego tematu wraz z przykładami to [JARP01] i [BHAR00].

Przegląd architektury Itanium znajduje się w [SHAR00]; bardziej szczegółowe ujęcie można znaleźć w [INTE00b].

Zarówno [TRIE01], jak i [MARK00] zawierają bardziej szczegółowe ujęcie tematów tego rozdziału. Wreszcie wyczerpujące omówienie architektury IA-64 i listy rozkazów znajduje się w [INTE00a].

BHAR00 Bharandwaj J. et al.: „The Intel IA-64 Compiler Code Generator". *IEEE Micro*, September/October 2000.

CHAS00 Chasin A.: „Predication, Speculation and Modern CPUs". *Dr Dobb's Journal*, May 2000.

DULO98 Dulong C.: „The IA-64 Architecture at Work". *Computer*, July 1998.

HUCK00 Huck J. et al.: „Introducing to IA-64 Architecture". *IEEE Micro*, September/October 2000.

HWU98 Hwu W.: „Introduction to Predicated Execution". *Computer*, January 1998.

HWU01 Hwu W., August D., Sias J.: „Program Decision Logic Optimization Using Predication and Control Speculation". *Proceedings of the IEEE*, November 2001.

INTE00a Intel Corp.: *Intel IA-64 Architecture Software Developer's Manual (4 volumes)*. Document 245317 through 245320, Aurora CO, 2000.

INTE00b Intel Corp.: *Itanium Processor Microarchitecture Reference for Software Optimization*. Aurora CO, Document 245320, August 2000.

JARP01 Jarp S.: „Optimizing IA-64 Performance". *Dr Jobb's Journal*, July 2001.

KATH01 Kathail B., Schlansker M., Rau B.: „Compiling for EPIC Architecture". *Proceedings of the IEEE,* November 2001.

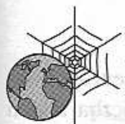
MARK00 Markstein P.: *IA-64 and Elementary Functions*. Upper Saddle River, Prentice Hall PTR, 2000.

SCHL00a Schlansker M., Rau B.: „EPIC: Explicitly Parallel Instruction Computing". *Computer,* February 2000.

SCHL00b Schlansker M., Rau B.: *EPIC: An Architecture for Instruction-Level Parallel Processors*. HPL Technical Report HPL-1999-111, Hewlett-Packard Laboratories (www.hpl.hp.com), February 2000.

SHAR00 Sharangpani H., Arona K.: „Itanium Processor Microarchitecture". *IEEE Micro*, September/October 2000.

TRIE01 Triebel W.: *Itanium Architecture for Software Developers*. Intel Press, 2001.



Polecane witryny WWW:

- **Itanium.** Witryna firmy Intel z najnowszymi informacjami o IA-64 i Itanium.
- **IMPACT.** Jest to witryna University of Illinois, gdzie przeprowadzono wiele badań na temat predykatywnego wykonywania rozkazów. Dostępnych jest tutaj wiele artykułów na ten temat.

15.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Architektura IA-64 – *IA-64 architecture*
Bardzo długie słowo rozkazu (VLIW) – *very long instruction word*
Bit NaT
Główny kod operacji – *major opcode*
Grupa rozkazów – *instruction group*
Itanium
Jawnie równoległe przetwarzanie rozkazów (EPIC) – *explicitly parallel instruction computing*
Jednostka wykonująca – *execution unit*
Ładowanie spekulatywne – *speculative loading*
Ładowanie wyprzedzające – *advanced load*
Pole szablonu – *template field*
Potok programowy – *software pipeline*
Predykacja – *predication*
Przerzucanie – *hoist*
Przewidywanie rozgałęzień – *branch predication*
Ramka stosu – *stack frame*
Rejestr predykacji – *predicate register*
Spekulacja danymi – *data speculation*
Spekulacja sterowaniem – *control speculation*
Stos rejestrowy – *register stack*
Sylaba – *syllable*
Układ uzupełniania rozkazu – *instruction completer*
Wiązka – *bundle*
Zatrzymanie – *stop*

Pytania kontrolne

15.1. Jakie są rodzaje jednostek wykonujących w IA-64?
15.2. Objaśnij zastosowanie pola szablonu w wiązce IA-64.
15.3. Jakie jest znaczenie zatrzymania w strumieniu rozkazów?
15.4. Zdefiniuj predykację i wykonywanie predykatywne.
15.5. W jaki sposób predykaty mogą zastąpić rozkaz rozgałęzienia warunkowego?
15.6. Zdefiniuj spekulowanie sterowaniem.
15.7. Do czego służy bit NaT?
15.8. Zdefiniuj spekulowanie danymi.
15.9. Jaka jest różnica między potokiem sprzętowym a programowym?
15.10. Jaka jest różnica między rejestrami tworzącymi stos a rotacyjnymi?

Problemy do rozwiązania

15.1. Załóżmy, że kod operacji IA-64 akceptuje trzy rejestry jako źródła argumentów i wytwarza wynik mieszczący się w jednym rejestrze. Jaka jest maksymalna liczba takich kodów operacji, która może być zdefiniowana w ramach jednej rodziny głównych kodów operacji?

15.2. W pewnym punkcie programu IA-64 występuje 10 rozkazów typu A i 6 rozkazów zmiennopozycyjnych, które mogą być wydane jednocześnie. Ile sylab może się ukazać bez jakichkolwiek zatrzymań między nimi?

15.3. W problemie 15.2:

(a) Ile cykli jest wymaganych w przypadku niewielkiej implementacji IA-64 zawierającej jedną jednostkę zmiennopozycyjną, dwie jednostki całkowitoliczbowe i dwie jednostki pamięci?

(b) Ile cykli jest wymaganych w przypadku organizacji Itanium pokazanej na rys. 15.10?

15.4. Algorytm, który może wykorzystywać cztery rozkazy zmiennopozycyjne na cykl, jest kodowany dla IA-64. Czy grupa rozkazów powinna zawierać cztery operacje zmiennopozycyjne? Jakie byłyby konsekwencje tego, gdyby maszyna realizująca ten program miała mniej niż cztery jednostki zmiennopozycyjne?

15.5. W podrozdziale 15.3 zostały przedstawione następujące rozkazy przeznaczone do wykonywania predykatywnego:

```
     cmp.crel p2, p3 = a, b
(p1) cmp.crel p2, p3 = a, b
```

gdzie `crel` jest relacją taką jak `eq`, `ne` itp.; p1, p2 i p3 to rejestry predykacji; a jest albo rejestrem, albo argumentem natychmiastowym; b jest argumentem rejestrowym. Wypełnij następującą tablicę prawdy:

p1	porównanie	p2	p3
brak	0		
brak	1		
0	0		
0	1		
1	0		
1	1		

15.6. W odniesieniu do programu predykatywnego z podrozdz. 15.3, który stanowi implementację sieci czynności z rys. 15.4, wskaż:

(a) Rozkazy, które mogą być wykonywane równolegle.

(b) Rozkazy, które mogą być włączone do tej samej wiązki rozkazów IA-64.

15.7. Rozważ następujący segment kodu źródłowego:

```
for ( i = 0; i < 100; i++ )
    if (A[i] < 50 )
         j = j + 1;
    else
         k = k + 1;
```

(a) Napisz odpowiedni segment kodu asemblerowego Pentium.

(b) Napisz zmodyfikowany segment kodu asemblerowego IA-64, posługując się metodami wykonywania predykatywnego.

15.8. Rozważ następujący fragment programu C operujący wartościami zmiennopozycyjnymi:

```
a[i] = p * q;
c = a[j];
```

Kompilator nie jest w stanie ustalić, czy $i \neq j$, ma jednak powody sądzić, że tak prawdopodobnie jest.

(a) Napisz program IA-64 stanowiący implementację tego programu C, korzystając z ładowania wyprzedzającego. *Wskazówka*: rozkazami ładowania zmiennopozycyjnego i mnożenia są odpowiednio `ldf` i `fmpy`.

(b) Zmodyfikuj ten program, korzystając z predykacji zamiast ładowania wyprzedzającego.

(c) Jakie są zalety i wady obydwu tych podejść?

15.9. Załóżmy, że jest tworzona ramka stosu rejestrowego o rozmiarze równym SOF = 48. Jeśli rozmiar grupy rejestrów lokalnych wynosi SOL = 16:

(a) Ile jest rejestrów wyjściowych (SOO)?

(b) Które rejestry znajdują się w grupach rejestrów lokalnych i wyjściowych?

Część czwarta
Jednostka sterująca

Problemy przedstawione w części czwartej

W części trzeciej skupiliśmy się na rozkazach maszynowych i operacjach realizowanych przez procesor podczas wykonywania tych rozkazów. Poza zasięgiem tych rozważań znalazł się szczegółowy sposób, w jaki powoduje się wykonywanie poszczególnych operacji. Jest to rola jednostki sterującej.

Jednostka sterująca jest tą częścią procesora, która w istocie sprawia, że są wykonywane określone czynności. Wysyła ona sygnały sterujące na zewnątrz procesora w celu uruchomienia wymiany danych z pamięcią i z modułami wejścia-wyjścia. Wysyła również wewnętrzne sygnały sterujące, aby spowodować przenoszenie danych między rejestrami, wykonanie określonej funkcji przez jednostkę arytmetyczno-logiczną, a także w celu sterowania innymi operacjami wewnętrznymi. Sygnały wejściowe do jednostki sterującej pochodzą z rejestrów rozkazów oraz składają się również ze znaczników (flag) i sygnałów sterujących ze źródeł zewnętrznych (np. sygnałów przerwań).

Tematyka poszczególnych rozdziałów części czwartej

Rozdział 16. Działanie jednostki sterującej

W rozdziale 16 omówimy sposób wykonywania poszczególnych funkcji przez procesor, a w szczególności sposób sterowania różnymi elementami procesora przez jednostkę sterującą, zapewniający realizowanie tych funkcji. Pokażemy, że każdy cykl rozkazu składa się ze zbioru mikrooperacji, które generują sygnały sterowania. Wykonywanie rozkazów zachodzi w wyniku tych sygnałów sterujących, które są przekazywane z jednostki sterującej do jednostki arytmetyczno-logicznej, rejestrów i do struktury połączeń systemowych. Zostanie również przedstawiona implementacja jednostki sterującej określana jako układowa (*hardwired*).

Rozdział 17. Sterowanie mikroprogramowe

W rozdziale 17 dowiemy się, w jaki sposób koncepcja mikrooperacji doprowadziła do eleganckiej i wydajnej implementacji jednostki sterującej w postaci mikroprogramowania. W istocie powstał język programowania niższego poziomu. Każdy rozkaz w języku maszynowym procesora jest tłumaczony na sekwencję rozkazów niższego poziomu, adresowanych do jednostki sterującej. Te rozkazy niższego poziomu są nazywane mikrorozkazami, a sam proces translacji jest określany jako mikroprogramowanie. Zostanie opisany układ pamięci sterującej zawierający mikroprogram dla każdego rozkazu maszynowego. Następnie zostanie wyjaśniona struktura i działanie mikroprogramowanej jednostki sterującej.

Rozdział 16

Działanie jednostki sterującej

16.1. Mikrooperacje
Cykl pobierania
Cykl adresowania pośredniego
Cykl przerwania
Cykl wykonywania
Cykl rozkazu
16.2. Sterowanie procesorem
Wymagania funkcjonalne
Sygnały sterujące
Przykład sygnałów sterujących
Wewnętrzna organizacja procesora
Procesor Intel 8085
16.3. Implementacja układowa
Wejścia jednostki sterującej
Rozwiązania logiczne jednostki sterującej
16.4. Polecana literatura
16.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

Podstawowe spostrzeżenia

- Wykonanie rozkazu obejmuje wykonanie szeregu kroków zwanych ogólnie *cyklami*. Na przykład może się ono składać z cykli pobrania, adresowania pośredniego, wykonywania i przerwania. Z kolei każdy cykl składa się z sekwencji operacji niższego rzędu, zwanych *mikrooperacjami*. Pojedyncza mikrooperacja polega na ogół na przeniesieniu danych między rejestrami, między rejestrami a magistralą zewnętrzną lub na wykonaniu prostej operacji przez ALU.
- Jednostka sterująca procesora realizuje dwa zadania: (1) powoduje wykonanie mikrooperacji przez procesor we właściwej kolejności wynikającej z realizowanego programu oraz (2) generuje sygnały sterujące, które powodują wykonywanie mikrooperacji.
- Sygnały sterujące generowane przez jednostkę sterującą powodują otwieranie i zamykanie bramek logicznych, co z kolei powoduje przenoszenie danych do (i z) rejestrów oraz wykonywanie operacji przez ALU.
- Jedna z metod implementacji jednostki sterującej – określana jako układowa – polega na tym, że jednostka ta jest traktowana jako układ kombinacyjny. Wejściowe sygnały logiczne wynikające z bieżącego rozkazu maszynowego są przetwarzane na zbiór wyjściowych sygnałów sterujących.

W rozdziale 10 stwierdziliśmy, że lista rozkazów maszynowych w znacznym stopniu określa procesor. Jeśli znamy listę rozkazów maszynowych, rozumiemy wyniki każdego kodu operacji, rozumiemy tryby adresowania i znamy zbiór rejestrów widzialnych dla użytkownika, znamy tym samym funkcje, które musi realizować procesor. Nie jest to jeszcze obraz całkowity. Musimy znać interfejsy zewnętrzne, dostępne zwykle za pośrednictwem magistrali, oraz sposób przetwarzania przerwań. Zgodnie z tym sposobem rozumowania, do określenia działania procesora muszą być sprecyzowane:

1. Operacje (kody operacji).
2. Tryby adresowania.
3. Rejestry.
4. Interfejs modułu wejścia-wyjścia.
5. Interfejs modułu pamięci.
6. Struktura przetwarzania przerwań.

Lista ta, chociaż ogólna, jest raczej kompletna. Punkty od 1 do 3 są definiowane poprzez określenie listy rozkazów. Punkty 4 i 5 są zwykle definiowane przez określenie magistrali systemowej. Punkt 6 jest definiowany częściowo przez magistralę systemową, a częściowo przez sposób, w jaki procesor wspiera system operacyjny.

Powyższa lista może definiować wymagania funkcjonalne w stosunku do procesora. Poszczególne punkty określają, co musi robić procesor. Tego właśnie dotyczyły części II i III. Teraz zajmiemy się problemem, jak funkcje te są wykonywane

lub – dokładniej – w jaki sposób steruje się różnymi elementami procesora w celu wykonywania tych funkcji. Tak więc przechodzimy do analizowania jednostki sterującej, która kontroluje działanie procesora.

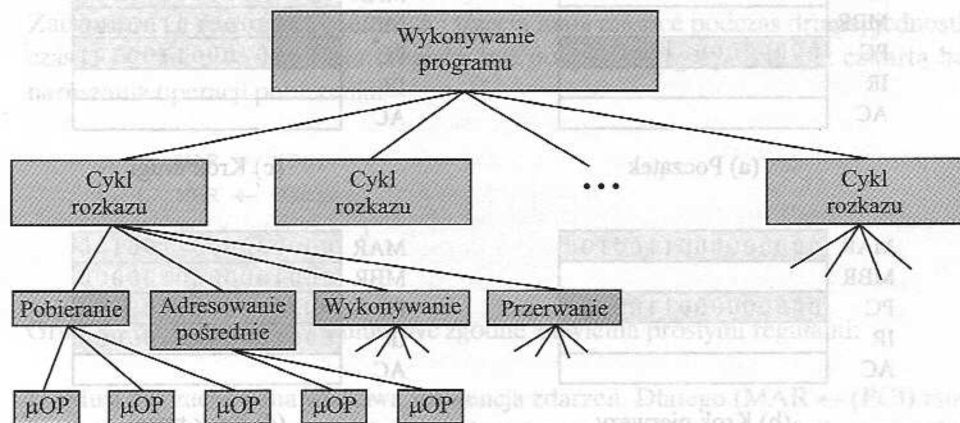
16.1. Mikrooperacje

Widzieliśmy, że działanie komputera podczas wykonywania programów składa się z sekwencji cykli rozkazu, przy czym jeden rozkaz maszynowy przypada na jeden cykl. Musimy oczywiście pamiętać, że ta sekwencja cykli rozkazu niekoniecznie jest taka sama, jak *napisana sekwencja rozkazów*, która tworzy program, ponieważ występują rozkazy rozgałęziania. Odnosimy się tutaj do wykonywania *czasowej sekwencji rozkazów*.

Stwierdziliśmy następnie, że każdy cykl rozkazu może być traktowany jako złożony z pewnej liczby mniejszych jednostek. Jednym z wygodnych podziałów jest podział na pobranie, adresowanie pośrednie, wykonanie i przerwanie, przy czym tylko cykle pobrania i wykonania występują zawsze.

W celu zaprojektowania jednostki sterującej musimy jednak podać bardziej szczegółowy opis. W ramach analizy przetwarzania potokowego w rozdz. 12 stwierdziliśmy już, że dalsza dekompozycja jest możliwa. Zobaczymy, że każdy z mniejszych cykli obejmuje szereg kroków, z których każdy angażuje rejestry procesora. Będziemy określać te kroki jako *mikrooperacje*. Przedrostek *mikro*- odnosi się do tego, że każdy krok jest bardzo prosty i osiąga bardzo niewiele. Na rysunku 16.1 są pokazane zależności między różnymi przedyskutowanymi dotychczas koncepcjami.

Podsumowując, wykonywanie programu polega na sekwencyjnym wykonywaniu rozkazów. Każdy rozkaz jest wykonywany podczas cyklu rozkazu złożonego z krótszych podcykli (np. pobrania, adresowania pośredniego, wykonania, przerwania). Realizacja każdego podcyklu obejmuje jedną lub więcej krótszych operacji, tzn. mikrooperacji.



Rysunek 16.1. Elementy składowe wykonywania programu (μOP – mikrooperacja)

Mikrooperacje są elementarnymi operacjami wykonywanymi przez procesor. W tym podrozdziale przeanalizujemy mikrooperacje, aby lepiej zrozumieć, w jaki sposób zdarzenia składające się na cykl rozkazu mogą być opisane jako sekwencja takich mikrooperacji. Wykorzystamy w tym celu prosty przykład. W pozostałej części rozdziału pokażemy, w jaki sposób koncepcja mikrooperacji może posłużyć jako przewodnik do projektowania jednostki sterującej.

Cykl pobierania

Rozpoczniemy analizę od cyklu pobierania, który występuje na początku każdego cyklu rozkazu i w którym następuje pobranie rozkazu z pamięci. Do celów dyskusji przyjmijmy schemat pokazany na rys. 12.6. Występują tam cztery rejestry:

- **Rejestr adresowy pamięci (MAR).** Jest on podłączony do linii adresu magistrali systemowej. Określa adres w pamięci dla operacji odczytu lub zapisu.
- **Rejestr buforowy pamięci (MBR).** Jest on podłączony do linii danych magistrali systemowej. Zawiera wartość, która ma być zapisana w pamięci lub która została ostatnio odczytana z pamięci.
- **Licznik programu (PC).** Zawiera adres następnego rozkazu, który ma być wykonywany.
- **Rejestr rozkazu (IR).** Zawiera ostatnio pobrany rozkaz.

Spójrzmy na sekwencję zdarzeń cyklu pobierania pod względem jej wpływu na rejestry procesora. Przykład widać na rys. 16.2. Na początku cyklu pobierania adres następnego rozkazu przewidzianego do wykonania znajduje się w liczniku programu (PC); w tym przypadku adresem jest 1100100. Pierwszym krokiem jest przeniesienie tego adresu do rejestru adresowego pamięci (MAR), ponieważ jest to jedyny rejestr podłączony do linii adresu magistrali systemowej. Żądany adres (zawarty w MAR) jest umieszczany na magistrali adresu, jednostka sterująca wydaje rozkaz READ (czytaj)

	(a) Początek
MAR	
MBR	
PC	0000000001100100
IR	
AC	

	(c) Krok drugi
MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100101
IR	
AC	

	(b) Krok pierwszy
MAR	0000000001100100
MBR	
PC	0000000001100100
IR	
AC	

	(d) Krok trzeci
MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100100
IR	0001000000100000
AC	

Rysunek 16.2. Sekwencja zdarzeń w cyklu pobierania

za pomocą szyny sterowania, wynik pojawia się na szynie danych i jest kopiowany do rejestru buforowego pamięci (MBR). Musi także nastąpić inkrementacja rejestru PC w celu przygotowania się do pobrania następnego rozkazu. Ponieważ te dwa działania (odczyt słowa z pamięci i dodanie 1 do PC) nie kolidują ze sobą, w celu zaoszczędzenia czasu możemy je wykonywać jednocześnie. Trzecim krokiem jest przeniesienie zawartości MBR do rejestru rozkazu (IR). Uwalnia to MBR, umożliwiając jego użycie w ewentualnym cyklu adresowania pośredniego.

W rzeczywistości prosty cykl pobierania składa się więc z 3 kroków i z 4 mikrooperacji. Każda mikrooperacja obejmuje przenoszenie danych do rejestru lub z rejestru. Tak długo, jak te przesunięcia danych nie kolidują ze sobą, możliwe jest oszczędzanie czasu przez wykonywanie ich w jednym kroku. Symbolicznie możemy zapisać tę sekwencję zdarzeń następująco:

t_1 : MAR ← (PC)
t_2 : MBR ← Pamięć
PC ← (PC) + I
t_3 : IR ← (MBR)

gdzie I jest długością rozkazu.

Koniecznych jest kilka komentarzy dotyczących tej sekwencji. Zakładamy, że do taktowania jest dostępny zegar wysyłający regularne impulsy. Każdy impuls zegara określa jednostkę czasu. Wobec tego wszystkie jednostki czasu mają tę samą długość. Każda mikrooperacja może być wykonana w ciągu jednej jednostki czasu. Zapisy (t_1, t_2, t_3) reprezentują kolejne jednostki czasu. Można to wyrazić słownie:

- **Pierwsza jednostka czasu** (t_1). Przeniesienie zawartości PC do MAR.
- **Druga jednostka czasu** (t_2). Przeniesienie zawartości lokacji pamięci określonej w MAR do MBR. Zwiększenie zawartości PC o I.
- **Trzecia jednostka czasu** (t_3). Przeniesienie zawartości MBR do IR.

Zauważmy, że mikrooperacje druga i trzecia mają miejsce podczas drugiej jednostki czasu. Trzecia mikrooperacja mogłaby być połączona (zgrupowana) z czwartą bez naruszania operacji pobierania:

t_1 : MAR ← (PC)
t_2 : MBR ← Pamięć
t_3 : PC ← (PC) + I
IR ← (MBR)

Grupowanie mikrooperacji musi być zgodne z dwiema prostymi regułami:

1. Musi być zachowana właściwa sekwencja zdarzeń. Dlatego (MAR ← (PC)) musi poprzedzać (MBR ← Pamięć), ponieważ operacja odczytu pamięci używa adresu zawartego w MAR.

2. Nie można dopuścić do konfliktów. Nie powinno się próbować odczytu i zapisu w tym samym rejestrze w tej samej jednostce czasu, ponieważ wyniki mogą być nieprzewidywalne. Na przykład mikrooperacje (MBR ← Pamięć) i (IR ← MBR) nie powinny następować w tej samej jednostce czasu.

Na zakończenie warto zauważyć, że jedna z mikrooperacji zawiera dodawanie. Aby uniknąć dublowania układów, dodawanie to mogłoby być wykonywane przez ALU. Jednak użycie ALU może spowodować wprowadzenie dodatkowych mikrooperacji, zależnie od funkcjonalności ALU i organizacji procesora. Odłożymy dyskusję na ten temat do dalszej części rozdziału.

Pożyteczne jest porównanie zdarzeń opisanych w tym i w następnym punkcie z rys. 3.5. Na tym rysunku mikrooperacje zostały zignorowane, nasza dyskusja wskazuje mikrooperacje potrzebne do przeprowadzenia podcykli składających się na cykl rozkazu.

Cykl adresowania pośredniego

Gdy rozkaz został już pobrany, następnym krokiem jest pobranie argumentów źródłowych. Kontynuując nasz prosty przykład, załóżmy jednoadresowy format rozkazu z dozwolonym adresowaniem bezpośrednim i pośrednim. Jeśli rozkaz określa adres pośredni, to cykl wykonywania musi być poprzedzony cyklem adresowania pośredniego. Przepływ danych jest pokazany na rys. 12.7 i obejmuje następujące mikrooperacje:

```
t₁ : MAR ← (IR(Adres))
t₂ : MBR ← Pamięć
t₃ : IR(Adres) ← (MBR(Adres))
```

Zawartość pola adresowego rozkazu jest przenoszona do rejestru MAR. Następnie jest używana do pobrania adresu argumentu. Na zakończenie pole adresu IR jest aktualizowane za pomocą zawartości rejestru MBR, dzięki czemu obecnie zawiera adres bezpośredni zamiast pośredniego.

Stan rejestru IR jest obecnie taki sam, jak gdyby adresowanie pośrednie nie było stosowane, i jest on gotowy do cyklu wykonywania. Pominiemy chwilowo ten cykl i rozpatrzmy cykl przerwania.

Cykl przerwania

Na zakończenie cyklu wykonywania jest przeprowadzany test mający na celu sprawdzenie, czy nie wystąpiły jakiekolwiek dozwolone przerwania. Jeśli wystąpiły, to następuje cykl przerwania. Natura tego cyklu różni się znacznie w zależności od komputera. Przedstawiamy bardzo prostą sekwencję zdarzeń zilustrowaną na rys. 12.8. Mamy:

```
t₁ : MBR ← (PC)
t₂ : MAR ← Adres zapisywany
     PC ← Adres programu obsługi
t₃ : Pamięć ← (MBR)
```

W ramach pierwszego kroku zawartość licznika PC jest przenoszona do rejestru MBR, dzięki czemu może ona być zachowana do powrotu z przerwania. Następnie do rejestru MAR jest ładowany adres, pod którym ma być zachowana zawartość PC, a do PC jest ładowany adres początkowy programu obsługi przerwań. Każde z tych dwóch działań może być pojedynczą mikrooperacją. Ponieważ jednak większość procesorów umożliwia wiele rodzajów i (lub) poziomów przerwań, dostęp do obu adresów przed ich przeniesieniem do MAR i PC może wymagać jednej lub wielu dodatkowych mikrooperacji. W każdym przypadku, gdy jest to już wykonane, krokiem końcowym jest zapis w pamięci starej wartości PC zawartej w MBR. Procesor jest teraz gotowy do rozpoczęcia następnego cyklu rozkazu.

Cykl wykonywania

Cykle pobierania, adresowania pośredniego i przerwania są proste i przewidywalne. Każdy z nich obejmuje niewielką, ustaloną sekwencję mikrooperacji i w każdym przypadku są powtarzane te same mikrooperacje.

Nie dotyczy to cyklu wykonywania. W przypadku maszyny o *N* różnych kodach operacji może wystąpić *N* różnych sekwencji mikrooperacji. Opiszemy kilka hipotetycznych przykładów.

Najpierw rozważmy rozkaz dodawania:

```
ADD R1, X
```

który powoduje dodanie zawartości lokacji X do rejestru R1. Podczas wykonywania tego rozkazu mogłaby wystąpić następująca sekwencja mikrooperacji:

t_1 : MAR ← (IR(Adres))
t_2 : MBR ← Pamięć
t_3 : R1 ← (R1) + (MBR)

Na początku rejestr IR zawiera rozkaz dodawania. Podczas pierwszego kroku część adresowa rejestru IR jest ładowana do rejestru MAR. Następnie jest odczytywana odpowiednia lokacja w pamięci. Na zakończenie ALU dodaje zawartości R1 i MBR. Mogą być wymagane dodatkowe mikrooperacje w celu wydobycia odniesienia do rejestru z rejestru IR i być może w celu zatrzymania danych wejściowych i wyjściowych ALU w pewnych pośrednich rejestrach.

Zapoznajmy się teraz z dwoma bardziej złożonymi przykładami. Powszechnie spotykanym rozkazem jest rozkaz inkrementacji i pominięcia w przypadku zera (*increment and skip if zero*):

```
ISZ X
```

Zawartość lokacji X jest zwiększana o 1. Jeśli wynikiem jest 0, następny rozkaz jest pomijany. Możliwa jest następująca sekwencja mikrooperacji:

```
t1 : MAR ← (IR(Adres))
t2 : MBR ← Pamięć
t3 : MBR ← (MBR) + 1
t4 : Pamięć ← (MBR)
     Jeśli (MBR = 0) to (PC ← (PC) + I)
```

Nową wprowadzoną tutaj właściwością jest działanie warunkowe. Stan licznika PC jest zwiększany, jeśli MBR = 0. Ten test oraz towarzyszące mu działanie mogą być wdrożone w postaci jednej mikrooperacji. Zauważmy także, że ta mikrooperacja może być przeprowadzona w ciągu tej samej jednostki czasu, podczas której zaktualizowana zawartość rejestru MBR jest zapisywana w pamięci.

Na zakończenie rozważmy rozkaz wywołania podprogramu. Jako przykład posłuży nam rozkaz rozgałęziania i zapisania adresu (*branch-and-save-address*):

```
BSA X
```

Adres rozkazu, który następuje po rozkazie BSA, jest zachowywany w lokacji X, a wykonywanie jest kontynuowane począwszy od lokacji X + 1. Zapisany adres będzie następnie użyty do powrotu. Jest to prosta metoda wywoływania podprogramów. Wystarcza następująca sekwencja mikrooperacji:

```
t1 : MAR ← (IR(Adres))
     MBR ← (PC)
t2 : PC ← (IR(Adres))
     Pamięć ← (MBR)
t3 : PC ← (PC) + I
```

Adres znajdujący się w liczniku PC na początku rozkazu jest adresem następnego rozkazu w sekwencji. Jest on zapisywany pod adresem wyznaczonym przez rejestr IR. Następuje również zwiększenie tego ostatniego adresu w celu określenia adresu rozkazu związanego z następnym cyklem rozkazu.

Cykl rozkazu

Stwierdziliśmy, że każda faza cyklu rozkazu może być traktowana jako sekwencja elementarnych mikrooperacji. W naszym przykładzie występuje po jednej sekwencji na cykle pobierania, adresowania pośredniego i przerwania, oraz w odniesieniu do cyklu wykonywania – po jednej sekwencji mikrooperacji dla każdego kodu operacji.

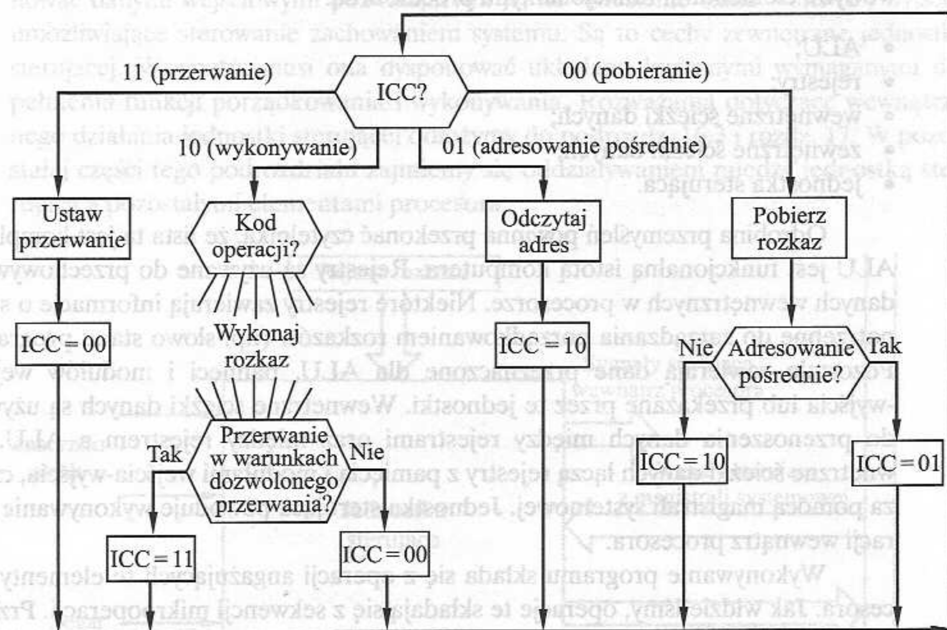
W celu uzupełnienia obrazu musimy powiązać ze sobą te sekwencje mikrooperacji, co zostało pokazane na rys. 16.3. Zakładamy, że istnieje nowy, 2-bitowy rejestr nazywany kodem cyklu rozkazu (*instruction cycle code* – ICC). Rejestr ICC wyznacza stan procesora, określając, w której części cyklu się on znajduje:

00: Pobieranie
01: Adresowanie pośrednie

10: Wykonywanie
11: Przerwanie

Na końcu każdego z czterech cykli następuje odpowiednie ustawienie rejestru ICC. Po cyklu adresowania pośredniego następuje zawsze cykl wykonywania. Po cyklu przerwania następuje zawsze cykl pobierania (patrz rys. 12.4). W przypadku cykli wykonywania i pobierania następny cykl zależy od stanu systemu.

Sieć działań na rys. 16.3 określa więc całkowitą sekwencję mikrooperacji, zależną jedynie od sekwencji rozkazów i przebiegu przerwań. Oczywiście jest to przykład uproszczony. Sieć działań dla rzeczywistego procesora byłaby bardziej złożona. W naszej dyskusji osiągnęliśmy jednak taki stan, w którym działanie procesora jest określane jako wykonywanie sekwencji mikrooperacji. Rozpatrzymy teraz, w jaki sposób jednostka sterująca powoduje realizowanie tej sekwencji.



Rysunek 16.3. Sieć czynności cyklu rozkazu

16.2. Sterowanie procesorem

Wymagania funkcjonalne

W wyniku analizy dokonanej w poprzednim podrozdziale zdekomponowaliśmy zachowanie (lub funkcjonowanie) procesora na operacje elementarne, zwane mikrooperacjami. Powodem naszego postępowania była chęć określenia natury jednostki sterującej. Przez zredukowanie działania procesora do najbardziej podstawowego

poziomu, jesteśmy w stanie dokładnie zdefiniować, jakie działania musi powodować jednostka sterująca. Możemy więc określić *wymagania funkcjonalne* odnoszące się do jednostki sterującej, tj. te funkcje, które musi realizować jednostka sterująca. Określenie tych wymagań funkcjonalnych jest podstawą projektowania i wdrażania jednostki sterującej.

Na podstawie tych informacji możemy przeprowadzić trzyetapowy proces zmierzający do scharakteryzowania jednostki sterującej:

1. Określenie podstawowych elementów procesora.
2. Opisanie mikrooperacji wykonywanych przez procesor.
3. Określenie funkcji, które musi realizować jednostka sterująca w celu spowodowania przeprowadzenia tych mikrooperacji.

Mamy już za sobą kroki 1 i 2. Podsumujmy ich wyniki. Po pierwsze, podstawowymi elementami funkcjonalnymi procesora są:

- ALU;
- rejestry;
- wewnętrzne ścieżki danych;
- zewnętrzne ścieżki danych;
- jednostka sterująca.

Odrobina przemyśleń powinna przekonać czytelnika, że lista ta jest kompletna. ALU jest funkcjonalną istotą komputera. Rejestry są używane do przechowywania danych wewnętrznych w procesorze. Niektóre rejestry zawierają informacje o stanie potrzebne do zarządzania porządkowaniem rozkazów (np. słowo stanu programu). Pozostałe zawierają dane przeznaczone dla ALU, pamięci i modułów wejścia-wyjścia lub przekazane przez te jednostki. Wewnętrzne ścieżki danych są używane do przenoszenia danych między rejestrami oraz między rejestrem a ALU. Zewnętrzne ścieżki danych łączą rejestry z pamięcią i modułami wejścia-wyjścia, często za pomocą magistrali systemowej. Jednostka sterująca powoduje wykonywanie operacji wewnątrz procesora.

Wykonywanie programu składa się z operacji angażujących te elementy procesora. Jak widzieliśmy, operacje te składają się z sekwencji mikrooperacji. Przeglądając podrozdz. 16.1, powinno się zauważyć, że wszystkie mikrooperacje dzielą się na następujące kategorie:

- ❑ Transfer danych z jednego rejestru do drugiego.
- ❑ Transfer danych z rejestru do interfejsu zewnętrznego (np. do magistrali systemowej).
- ❑ Transfer danych z interfejsu zewnętrznego do rejestru.
- ❑ Wykonywanie operacji arytmetycznej lub logicznej, przy czym do przechowania danych wejściowych i wyjściowych służą rejestry.

Wszystkie mikrooperacje wymagane do zrealizowania jednego cyklu rozkazu, łącznie ze wszystkimi mikrooperacjami potrzebnymi do wykonania każdego rozkazu z listy rozkazów, kwalifikują się do jednej z tych kategorii.

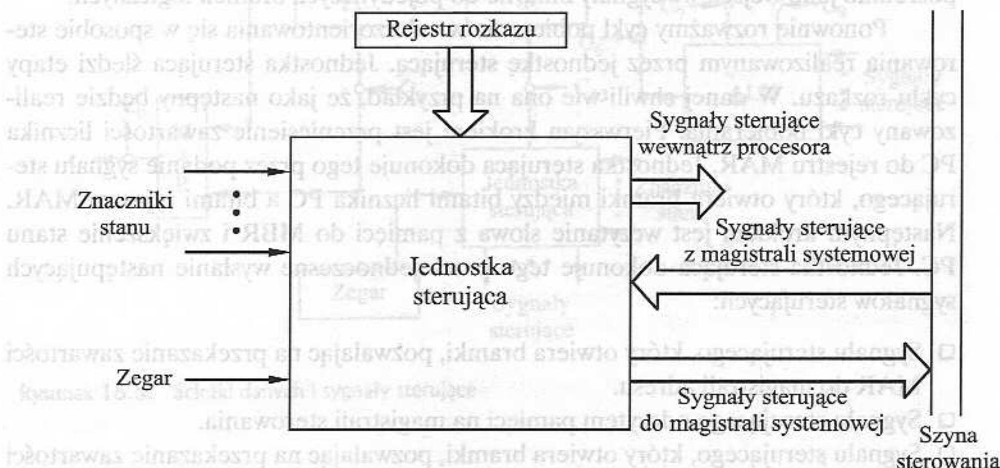
Możemy już teraz w pewnym stopniu wyjaśnić sposób funkcjonowania jednostki sterującej. Realizuje ona dwa podstawowe zadania:

- **Szeregowanie.** Jednostka sterująca powoduje, że procesor wykonuje ciąg mikrooperacji we właściwej kolejności (sekwencji) opartej na wykonywanym programie.
- **Wykonywanie.** Jednostka sterująca powoduje wykonanie każdej mikrooperacji.

Przedstawiliśmy opis funkcjonalny tego, co wykonuje jednostka sterująca. Kluczem do jej funkcjonowania jest użycie sygnałów sterujących.

Sygnały sterujące

Określiliśmy elementy procesora (ALU, rejestry, ścieżki danych) oraz wykonywane mikrooperacje. Aby jednostka sterująca mogła realizować swoje funkcje, musi dysponować danymi wejściowymi pozwalającymi jej na określenie stanu systemu i wyjścia umożliwiające sterowanie zachowaniem systemu. Są to cechy zewnętrzne jednostki sterującej. Wewnątrz musi ona dysponować układami logicznymi wymaganymi do pełnienia funkcji porządkowania i wykonywania. Rozważania dotyczące wewnętrznego działania jednostki sterującej odłożymy do podrozdz. 16.3 i rozdz. 17. W pozostałej części tego podrozdziału zajmiemy się oddziaływaniem między jednostką sterującą a pozostałymi elementami procesora.



Rysunek 16.4. Model jednostki sterującej

Na rysunku 16.4 jest pokazany ogólny model jednostki sterującej wraz ze wszystkimi wejściami i wyjściami. Źródłami danych wejściowych jednostki sterującej są:

- **Zegar.** Dzięki niemu działanie jednostki sterującej jest związane z przebiegiem czasu. Jednostka sterująca powoduje wykonywanie jednej mikrooperacji (lub zbioru jednoczesnych mikrooperacji) po każdym impulsie zegarowym. Jest to niekiedy określane jako czas cyklu procesora lub czas cyklu zegara.

- **Rejestr rozkazu.** Do określenia, które mikrooperacje mają być wykonane podczas cyklu wykonywania, jest używany kod operacji bieżącego rozkazu.
- **Znaczniki stanu.** Są one potrzebne jednostce sterującej do określania stanu procesora i wyników poprzednich operacji ALU. Na przykład, wykonując rozkaz inkrementacji i pominięcia w przypadku zera (ISZ), jednostka sterująca zwiększy stan licznika PC, jeśli jest ustawiony znacznik stanu 0.
- **Magistrala sterowania.** Magistrala sterowania stanowiąca część magistrali systemowej przekazuje sygnały jednostce sterującej, takie jak sygnały przerwania i potwierdzenia.

Natomiast na wyjściu pojawiają się:

- **Wewnętrzne sygnały sterujące procesora.** Istnieją dwa ich rodzaje: te, które powodują przenoszenie danych z jednego rejestru do drugiego, oraz te, które aktywują określone funkcje ALU.
- **Sygnały sterujące do magistrali sterowania.** Również istnieją dwa rodzaje tych sygnałów: sygnały sterujące pamięcią i sygnały sterujące modułami wejścia-wyjścia.

Nowym elementem wprowadzonym na rysunku jest sygnał sterujący. Używane są trzy rodzaje sygnałów sterujących: aktywujące działanie ALU, aktywujące ścieżkę danych oraz te, które znajdują się na zewnętrznej magistrali systemowej lub w innym interfejsie zewnętrznym. Wszystkie te sygnały są z zasady doprowadzane bezpośrednio jako wejściowe sygnały binarne do pojedynczych bramek logicznych.

Ponownie rozważmy cykl pobierania w celu zorientowania się w sposobie sterowania realizowanym przez jednostkę sterującą. Jednostka sterująca śledzi etapy cyklu rozkazu. W danej chwili wie ona na przykład, że jako następny będzie realizowany cykl pobierania. Pierwszym krokiem jest przeniesienie zawartości licznika PC do rejestru MAR. Jednostka sterująca dokonuje tego przez podanie sygnału sterującego, który otwiera bramki między bitami licznika PC a bitami rejestru MAR. Następnym krokiem jest wczytanie słowa z pamięci do MBR i zwiększenie stanu PC. Jednostka sterująca dokonuje tego przez jednoczesne wysłanie następujących sygnałów sterujących:

- Sygnału sterującego, który otwiera bramki, pozwalając na przekazanie zawartości MAR do magistrali adresu.
- Sygnału sterującego odczytem pamięci na magistrali sterowania.
- Sygnału sterującego, który otwiera bramki, pozwalając na przekazanie zawartości magistrali danych do rejestru MBR.
- Sygnału sterującego do układów logicznych, które dodają 1 do zawartości PC i zapisują wynik w PC.

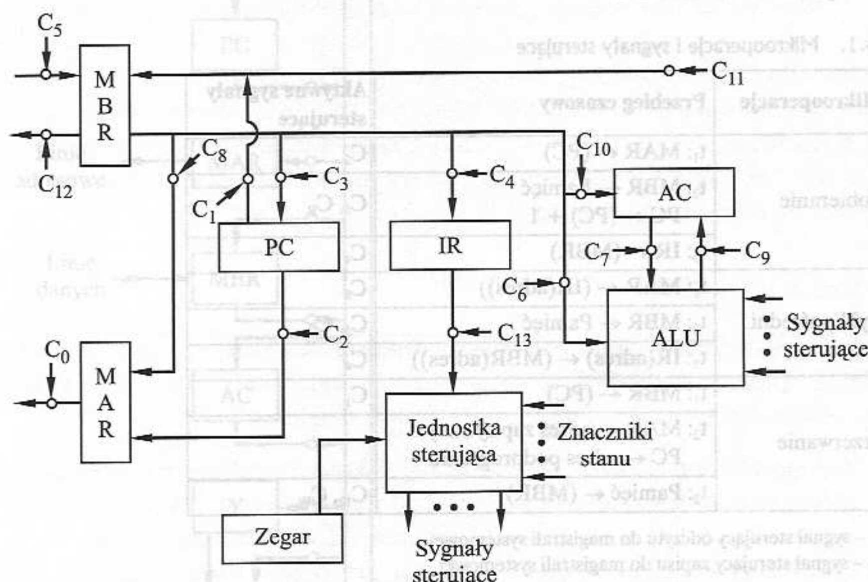
Następnie jednostka sterująca wysyła sygnał sterujący, który otwiera bramki między MBR a IR.

Kończy to cykl pobierania z wyjątkiem jednej rzeczy: jednostka sterująca musi zdecydować, czy jako następny ma być realizowany cykl adresowania pośredniego, czy cykl wykonywania. W celu podjęcia tej decyzji jednostka sterująca analizuje zawartość rejestru IR, aby stwierdzić, czy ma miejsce pośrednie odniesienie do pamięci.

Cykle adresowania pośredniego i przerwania są realizowane podobnie. W przypadku cyklu wykonywania jednostka sterująca rozpoczyna od analizy kodu operacji i na tej podstawie decyduje, która sekwencja mikrooperacji ma być realizowana w tym cyklu.

Przykład sygnałów sterujących

Aby zilustrować funkcjonowanie jednostki sterującej, przeanalizujmy prosty przykład. Został on zilustrowany na rys. 16.5. Jest to prosty procesor z pojedynczym akumulatorem. Na rysunku są pokazane ścieżki danych między elementami. Ścieżki sterowania sygnałów wysyłanych przez jednostkę sterującą nie zostały pokazane, jednak końcówki sygnałów sterowania oznaczono przez C_i i wyróżniono za pomocą



Rysunek 16.5. Ścieżki danych i sygnały sterujące

kółek. Jednostka sterująca otrzymuje sygnały wejściowe z zegara i z rejestru rozkazu oraz znaczniki stanu. W każdym cyklu zegara jednostka sterująca odczytuje wszystkie swoje wejścia i wysyła zbiór sygnałów sterujących. Sygnały sterujące są kierowane do trzech miejsc przeznaczenia:

- **Ścieżki danych.** Jednostka sterująca steruje wewnętrznym przepływem danych. Na przykład w czasie pobierania rozkazu zawartość rejestru buforowego pamięci jest przenoszona do rejestru rozkazu. W każdej kontrolowanej ścieżce istnieje bramka (reprezentowana przez kółko na rysunku). Sygnał sterujący z jednostki sterującej okresowo otwiera bramkę, umożliwiając przepływ danych.

- **ALU.** Jednostka sterująca kontroluje pracę ALU za pomocą zbioru sygnałów sterujących. Sygnały te aktywują różne elementy logiczne i bramki wewnątrz ALU.
- **Magistrala systemowa.** Jednostka sterująca wysyła sygnały sterujące na zewnątrz za pośrednictwem linii sterowania magistrali systemowej (np. przy odczycie pamięci).

Jednostka sterująca musi wciąż wiedzieć, jaki fragment cyklu rozkazu jest aktualnie realizowany. Dysponując tą wiedzą i odczytując wszystkie swoje wejścia, jednostka sterująca wysyła sekwencję sygnałów sterujących, które powodują wykonywanie mikrooperacji. Używa ona impulsów zegarowych do taktowania sekwencji zdarzeń, rezerwując czas między zdarzeniami na stabilizowanie się poziomów sygnałów. W tabeli 16.1 są podane sygnały sterujące potrzebne dla wcześniej opisanych sekwencji mikrooperacji. Dla uproszczenia pominięto ścieżki danych i sterowania służące do zwiększania stanu licznika PC i do ładowania ustalonych adresów do rejestrów PC i MAR.

Tabela 16.1. Mikrooperacje i sygnały sterujące

Mikrooperacje	Przebieg czasowy	Aktywne sygnały sterujące
Pobieranie	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Pamięć PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Cykl pośredni	t_1: MAR ← (IR(adres))	C_8
	t_2: MBR ← Pamięć	C_5, C_R
	t_3: IR(adres) ← (MBR(adres))	C_4
Przerwanie	t_1: MBR ← (PC)	C_1
	t_2: MAR ← adres zapisywany PC ← adres podprogramu	
	t_3: Pamięć ← (MBR)	C_{12}, C_W

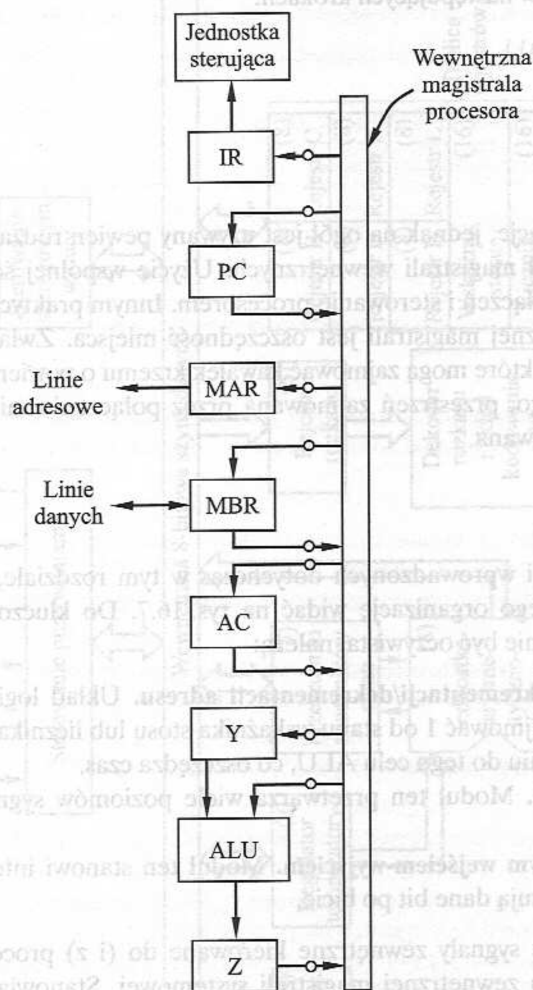
C_R – sygnał sterujący odczytu do magistrali systemowej
C_W – sygnał sterujący zapisu do magistrali systemowej

Godna zastanowienia jest minimalistyczna natura jednostki sterującej. Jednostka ta jest „silnikiem" napędzającym cały komputer. Dokonuje tego tylko na podstawie znajomości przewidzianych do wykonania rozkazów oraz natury wyników operacji arytmetycznych i logicznych (np. dodatni, przepełnienie itd.). Nigdy nie zna ona przetwarzanych danych lub rzeczywistych wyników. Steruje tym wszystkim za pomocą niewielu sygnałów sterujących kierowanych do pewnych punktów w procesorze oraz niewielu sygnałów kierowanych do magistrali systemowej.

Wewnętrzna organizacja procesora

Na rysunku 16.5 jest zilustrowane zastosowanie wielu różnych ścieżek danych. Złożoność tego typu organizacji jest oczywista. Częściej używana jest pewna odmiana wewnętrznej magistrali, co już pokazaliśmy na rys. 12.2.

Jeśli zastosuje się wewnętrzną magistralę procesora, to rys. 16.5 można przekształcić, otrzymując rys. 16.6. ALU i wszystkie rejestry procesora są połączone za pomocą jednej magistrali wewnętrznej. Przewidziane zostały bramki i sygnały sterujące, które kontrolują ruch danych między każdym rejestrem a magistralą. Dodatkowe sygnały sterujące kontrolują przesyłanie danych do (i z) magistrali systemowej (zewnętrznej) oraz działanie ALU.



Rysunek 16.6. Procesor z magistralą wewnętrzną

Do tej struktury dodano dwa nowe rejestry, oznaczone Y i Z. Są one potrzebne do właściwego działania ALU. Gdy jest wykonywana operacja wymagająca dwóch argumentów, jeden z nich może być uzyskany z magistrali wewnętrznej, jednak drugi musi pochodzić z innego źródła. Do tego celu mógłby być użyty akumulator (AC), jednak ograniczałoby to elastyczność systemu i nie działałoby w przypadku procesora z wieloma rejestrami roboczymi. Rejestr Y służy do czasowego przechowywania dru-

giego argumentu. ALU jest układem kombinacyjnym (patrz dodatek do książki) bez wewnętrznej pamięci. Wobec tego, gdy sygnały sterujące aktywują pewną funkcję ALU, dane wejściowe ALU są przekształcane i podawane na wyjście. Wyjście ALU nie może więc być bezpośrednio dołączone do magistrali, ponieważ istniałoby wówczas sprzężenie zwrotne wyjścia z wejściem. Rejestr Z umożliwia czasowe przechowywanie wartości wyjściowych. Przy takiej organizacji operacja dodawania wartości z pamięci do AC przebiegałaby w następujących krokach:

t_1 : MAR ← (IR(Adres))
t_2 : MBR ← Pamięć
t_3 : Y ← (MBR)
t_4 : Z ← (AC) + (Y)
t_5 : AC ← (Z)

Możliwe są inne organizacje, jednak na ogół jest używany pewien rodzaj magistrali wewnętrznej lub zespół magistrali wewnętrznych. Użycie wspólnej ścieżki danych upraszcza geometrię połączeń i sterowanie procesorem. Innym praktycznym powodem stosowania wewnętrznej magistrali jest oszczędność miejsca. Zwłaszcza w przypadku mikroprocesorów, które mogą zajmować kawałek krzemu o powierzchni zaledwie 1/4 cala kwadratowego, przestrzeń zajmowana przez połączenia międzyrejestrowe musi być minimalizowana.

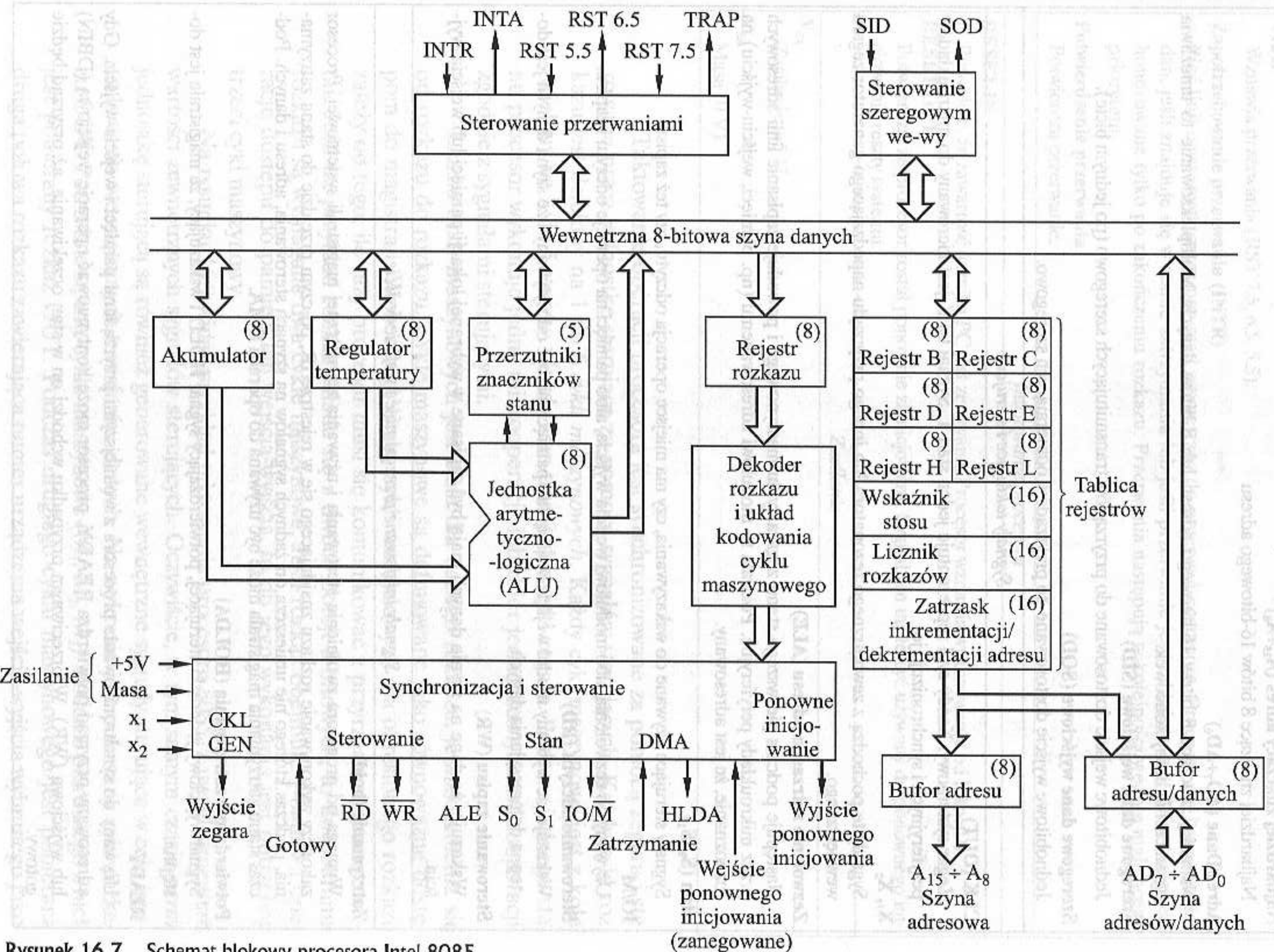
Procesor Intel 8085

W celu zilustrowania koncepcji wprowadzonych dotychczas w tym rozdziale, rozważmy procesor Intel 8085. Jego organizację widać na rys. 16.7. Do kluczowych składników, których rola może nie być oczywista, należą:

- **Przerzutnik zatrzaskowy inkrementacji/dekrementacji adresu.** Układ logiczny, który może dodawać lub odejmować 1 od stanu wskaźnika stosu lub licznika programu. Zapobiega to używaniu do tego celu ALU, co oszczędza czas.
- **Układ sterowania przerwań.** Moduł ten przetwarza wiele poziomów sygnałów przerwań.
- **Układ sterowania szeregowym wejściem-wyjściem.** Moduł ten stanowi interfejs z przyrządami, które przekazują dane bit po bicie.

W tabeli 16.2 są opisane sygnały zewnętrzne kierowane do (i z) procesora 8085. Są one wprowadzane do zewnętrznej magistrali systemowej. Stanowią one interfejs między procesorem 8085 a resztą systemu (rys. 16.8).

Pokazana jednostka sterująca składa się z dwóch modułów: (1) dekodera rozkazów i układu kodowania cyklu maszynowego oraz (2) układu taktowania i sterowania. Dyskusję na temat pierwszego modułu odłożymy do następnego podrozdziału. Jądrem jednostki sterującej jest moduł taktowania i sterowania. Zawiera on zegar; na jego wejście są podawane bieżący rozkaz i pewne zewnętrzne sygnały sterujące. Na wyjściu pojawiają się sygnały sterujące kierowane do innych składników CPU oraz sygnały sterujące kierowane do zewnętrznej magistrali systemowej.



Rysunek 16.7. Schemat blokowy procesora Intel 8085

Tabela 16.2. Sygnały zewnętrzne procesora Intel 8085

Sygnały adresów i danych
Najbardziej znaczący adres ($A_{15} \div A_8$) Najbardziej znaczące 8 bitów 16-bitowego adresu
Adres/Dane ($AD_7 \div AD_0$) Najmniej znaczące 8 bitów 16-bitowego adresu lub 8 bitów danych. Multipleksowanie to umożliwia ograniczenie liczby końcówek.
Szeregowe dane wejściowe (SID) Jednobitowe wejście dostosowane do przyrządów transmitujących szeregowo (po jednym bicie).
Szeregowe dane wyjściowe (SOD) Jednobitowe wyjście dostosowane do przyrządów odbierających szeregowo.
Sygnały taktujące i sterujące
CLK (OUT) Zegar systemowy. Każdy cykl reprezentuje jeden stan T. Sygnał CLK jest kierowany do mikroukładów peryferyjnych i synchronizuje je.
X_1, X_2 Sygnały te pochodzą z zewnętrznego rezonatora lub innego przyrządu napędzającego generator zegara wewnętrznego.
Zezwolenie zatrzasku adresu (ALE) Następuje podczas pierwszego stanu zegara cyklu maszynowego i powoduje zapisanie linii adresowych przez mikroukłady peryferyjne. Pozwala to modułowi adresowanemu (np. pamięci, wejściu-wyjściu) na rozpoznanie, że jest adresowany.
Stan (S_0, S_1) Sygnały sterujące używane do wskazywania, czy ma miejsce operacja odczytu, czy też zapisu.
IO/M Używany do zezwolenia albo modułowi wejścia-wyjścia, albo pamięci na operacje odczytu i zapisu.
Sterowanie odczytu (RD) Wskazuje, że wybrany moduł wejścia-wyjścia lub pamięci ma być odczytywany i że szyna danych jest dostępna do przesyłania danych.
Sterowanie zapisu (WR) Wskazuje, że dane na szynie danych mają być zapisane w wybranej lokacji pamięci lub wejścia-wyjścia.
Sygnały inicjowane przez pamięć i wejście-wyjście
Zatrzymanie (hold) Wymaga od procesora zwolnienia sterowania i używania zewnętrznej magistrali systemowej. Procesor zakończy wykonywanie rozkazu znajdującego się w rejestrze IR, po czym przejdzie do stanu zatrzymania, podczas którego nie umieszcza on żadnych sygnałów na szynach sterowania, adresu i danych. Podczas stanu zatrzymania magistrala może być używana do operacji DMA.
Potwierdzenie zatrzymania (HOLDA) Sygnał wyjściowy jednostki sterującej, potwierdzający sygnał HOLD i wskazujący, że magistrala jest dostępna.
READY Używany do synchronizowania procesora z wolniejszymi przyrządami pamięci i wejścia-wyjścia. Gdy adresowany przyrząd potwierdza READY, procesor może kontynuować operację wejściową (DBIN) lub wyjściową (WR). W przeciwnym przypadku wchodzi on w stan oczekiwania, aż przyrząd będzie gotowy.

Tabela 16.2. Sygnały zewnętrzne procesora Intel 8085 (cd.)

Sygnały związane z przerwaniami
TRAP Wznawia przerwania (RST 7.5, 6.5, 5.5). **Zapotrzebowanie przerwania (INTR)** Pięć linii używanych przez przyrząd zewnętrzny do przerwań procesora. Procesor nie honoruje żądań, jeśli znajduje się w stanie zatrzymania lub jeśli przerwanie jest zablokowane. Przerwanie jest honorowane tylko po zakończeniu rozkazu. Przerwania następują w kolejności według malejącego priorytetu. **Potwierdzenie przerwania** Potwierdza przerwanie.
Inicjowanie procesora
RESET IN Powoduje, że zawartość licznika PC jest zerowana. Procesor wznawia wykonywanie od lokacji 0. **RESET OUT** Potwierdza, że procesor został ponownie zainicjowany. Sygnał ten może być używany do ponownego inicjowania reszty systemu.
Zasilanie
V_{CC} Zasilanie +5 V. V_{SS} Masa (0 V).

Taktowanie operacji procesora jest synchronizowane za pomocą zegara i sterowane przez jednostkę sterującą za pomocą sygnałów sterujących. Każdy cykl rozkazu jest dzielony na 1 do 5 *cykli maszynowych*. Każdy cykl maszynowy jest z kolei dzielony na 3 do 5 *stanów*. Każdy stan trwa jeden cykl zegara. Podczas jednego stanu procesor wykonuje jedną mikrooperację lub zbiór jednoczesnych mikrooperacji, zgodnie z sygnałami sterującymi.

Liczba cykli maszynowych jest stała dla danego rozkazu, jednak zmienia się od rozkazu do rozkazu. Cykle maszynowe są definiowane jako równoważne dostępom do magistrali. Wobec tego liczba cykli maszynowych dla określonego rozkazu zależy od tego, ile razy procesor musi się komunikować z przyrządami zewnętrznymi. Jeśli na przykład rozkaz składa się z dwóch porcji 8-bitowych, to do pobrania rozkazu są wymagane dwa cykle maszynowe. Jeśli rozkaz obejmuje 1-bajtową operację dostępu do pamięci lub wejścia-wyjścia, to do jego wykonania jest potrzebny trzeci cykl maszynowy.

Na rysunku 16.9 widać przykład taktowania procesora 8085, z uwzględnieniem wartości zewnętrznych sygnałów sterujących. Oczywiście, w tym samym czasie przez jednostkę sterującą są również generowane wewnętrzne sygnały sterujące w celu kontrolowania wewnętrznych transferów danych. Na wykresie jest pokazany cykl rozkazu OUT. Potrzebne są trzy cykle maszynowe (M_1, M_2, M_3). Podczas pierwszego z nich jest pobierany rozkaz OUT. Podczas drugiego cyklu maszynowego jest pobierana druga połowa rozkazu zawierająca numer urządzenia wejścia-wyjścia wybranego jako

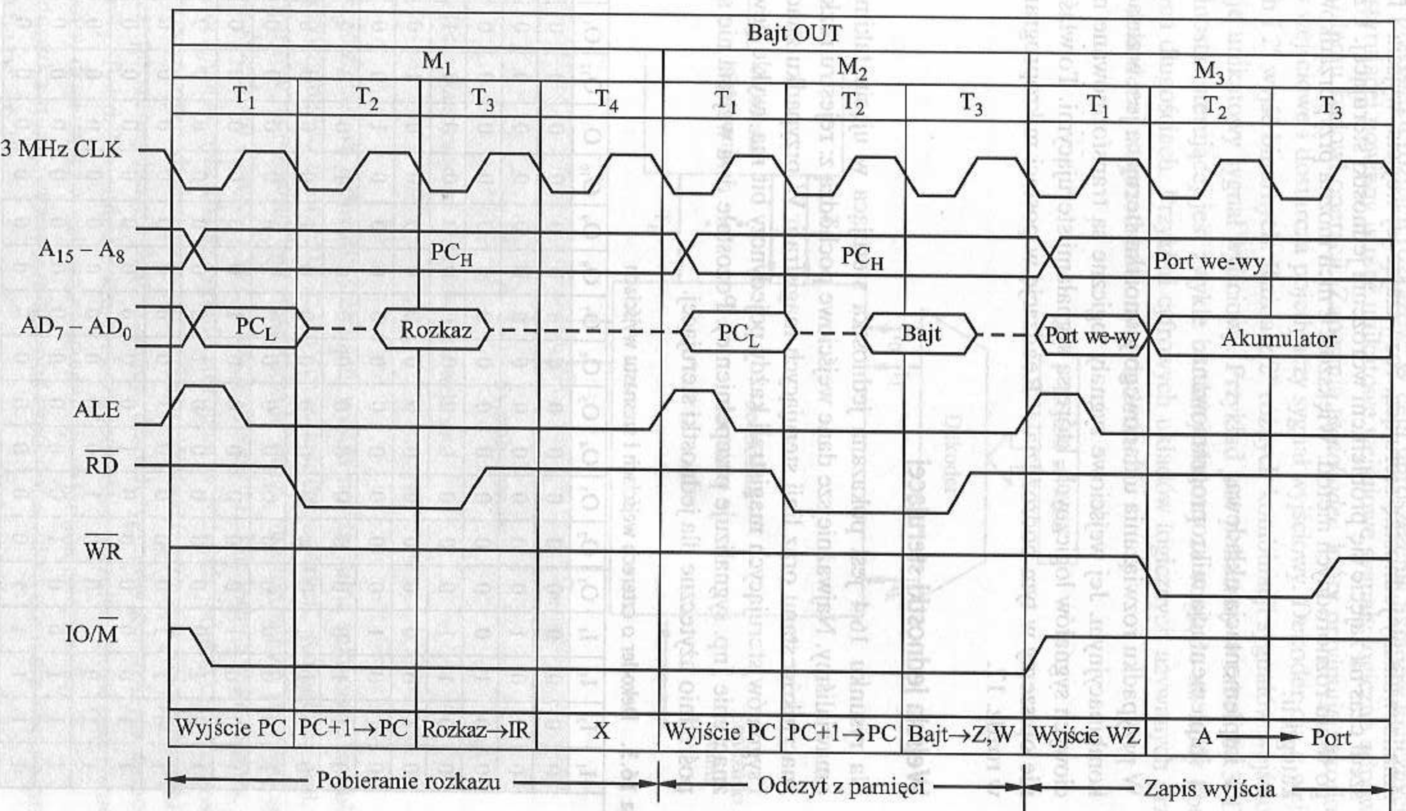
Sygnał	Nr	Nr	Sygnał
X_1	1	40	V_{CC}
X_2	2	39	HOLD
Wyjście ponownego inicjowania	3	38	HLDA
SOD	4	37	Wyjście zegara
SID	5	36	$\overline{\text{Wejście ponownego inicjowania (zanegowane)}}$
Pułapka	6	35	Gotowy
RST 7.5	7	34	IO/$\overline{M}$
RST 6.5	8	33	S_1
RST 5.5	9	32	$\overline{RD}$
$\overline{INTR}$	10	31	$\overline{WR}$
INTA	11	30	ALE
AD_0	12	29	S_0
AD_1	13	28	A_{15}
AD_2	14	27	A_{14}
AD_3	15	26	A_{13}
AD_4	16	25	A_{12}
AD_5	17	24	A_{11}
AD_6	18	23	A_{10}
AD_7	19	22	A_9
V_{ss}	20	21	A_8

Rysunek 16.8. Rozkład wyprowadzeń procesora Intel 8085

wyjściowe. Podczas trzeciego cyklu zawartość AC jest kierowana do wybranego urządzenia poprzez magistralę danych.

Początek każdego cyklu maszynowego jest sygnalizowany przez impuls zezwolenia przerzutnika zatrzaskowego adresu (*Address Latch Enabled* – ALE) pochodzący z jednostki sterującej. Impuls ALE wywołuje stan gotowości układów zewnętrznych. Podczas stanu T_1 cyklu maszynowego M_1 jednostka sterująca ustawia sygnał IO/M w celu pokazania, że jest to operacja pamięci. Jednostka sterująca powoduje również, że zawartość licznika PC jest umieszczana na szynie adresowej (A_{15}÷A_8) oraz na szynie adresów/danych (AD_7÷AD_0). Podczas opadającego zbocza impulsu ALE pozostałe moduły na magistrali zapisują adres.

Podczas stanu T_2 zaadresowany moduł pamięci umieszcza zawartość zaadresowanej lokacji na szynie adresów/danych. Jednostka sterująca ustawia sygnał sterowania odczytem (*Read Control* – RD) w celu zasygnalizowania odczytu, jednak skopiowanie danych z szyny jest opóźnione do T_3. Daje to modułowi pamięci czas na umieszczenie danych na magistrali oraz na ustabilizowanie poziomów sygnałów. Ostatni stan, T_4, jest stanem *nieaktywnej szyny*, podczas którego procesor dekoduje rozkaz. Pozostałe cykle maszynowe przebiegają podobnie.



Rysunek 16.9. Przebiegi czasowe rozkazu OUT procesora Intel 8085

16.3. Implementacja układowa

Analizowaliśmy jednostkę sterującą, rozważając jej wejścia, wyjścia i funkcje. Przyszedł czas na zajęcie się problemem wdrożenia jednostki sterującej. Używano do tego wielu różnorodnych metod. Większość z nich można przyporządkować do dwóch kategorii:

- implementacja układowa,
- implementacja mikroprogramowana.

W przypadku rozwiązania układowego jednostka sterująca jest w zasadzie układem kombinacyjnym. Jej wejściowe sygnały logiczne są transformowane na zbiór wyjściowych sygnałów logicznych, które są sygnałami sterującymi. To właśnie rozwiązanie opiszemy w tym podrozdziale. Realizację w postaci mikroprogramu omówimy w rozdz. 17.

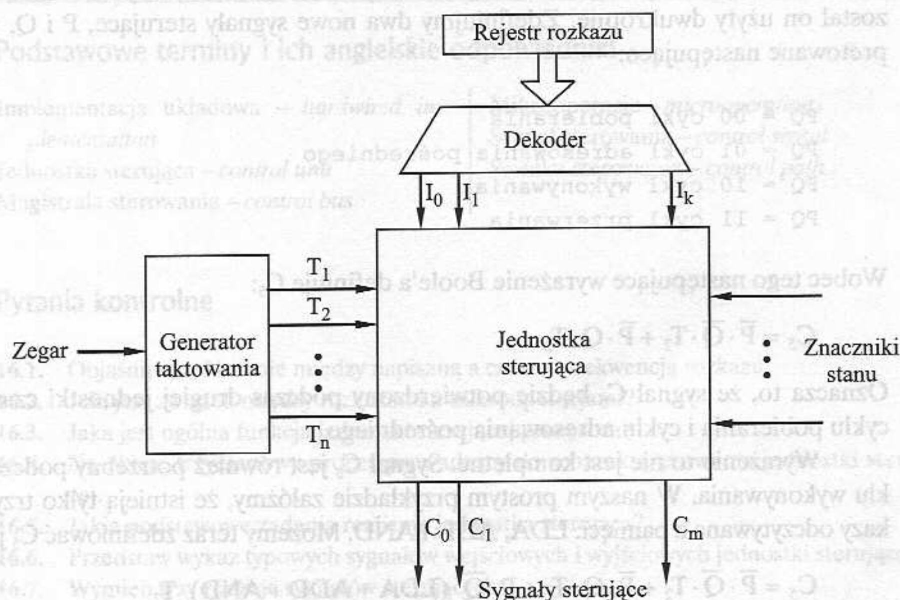
Wejścia jednostki sterującej

Na rysunku 16.4 jest pokazana jednostka sterująca w ujęciu takim, jakie dotąd omawialiśmy. Najważniejsze dane wejściowe pochodzą z: rejestru rozkazów, zegara, znaczników stanu oraz linii sterujących magistrali. W przypadku znaczników stanu i sygnałów sterujących magistrali każdy pojedynczy bit ma zwykle pewne określone znaczenie (np. sygnalizuje przepełnienie). Pozostałe dwa wejścia nie są jednak bezpośrednio użyteczne dla jednostki sterującej.

Tabela 16.3. Dekoder o czterech wejściach i szesnastu wyjściach

I_1	I_2	I_3	I_4	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}	O_{13}	O_{14}	O_{15}	O_{16}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Rozważmy najpierw rejestr rozkazów. Jednostka sterująca rozpoznaje kod operacji i wykonuje różne działania (generuje różne kombinacje sygnałów sterujących) w przypadku różnych rozkazów. W celu uproszczenia układów logicznych jednostki sterującej powinny istnieć unikatowe wejściowe sygnały logiczne odpowiadające każdemu kodowi operacji. Zapewnia to *dekoder*, który przyjmuje zakodowane sygnały wejściowe i dostarcza pojedynczy sygnał wyjściowy. Dekoder ma n wejść binarnych i 2^n wyjść binarnych. Każda z 2^n różnych kombinacji sygnałów wejściowych aktywuje unikatowy sygnał wyjściowy. Przykład jest pokazany w tabeli 16.3. Dekoder jednostki sterującej jest zwykle bardziej złożony ze względu na kody operacji o różnych długościach. Przykład cyfrowych układów logicznych używanych przy realizacji dekodera jest przedstawiony w dodatku A.



Rysunek 16.10. Jednostka sterująca ze zdekodowanymi wejściami

Część zegarowa jednostki sterującej wysyła powtarzalną sekwencję impulsów. Jest to przydatne do mierzenia czasu trwania mikrooperacji. Konieczne jest, aby okres impulsów zegarowych był dostatecznie długi, aby umożliwić propagację sygnałów w ścieżkach danych i w układach procesora. Jak jednak widzieliśmy, jednostka sterująca wysyła różne sygnały sterujące w różnych jednostkach czasowych w czasie trwania pojedynczego cyklu rozkazu. Wobec tego na wejściu jednostki sterującej jest potrzebny licznik dostarczający różnych sygnałów kontrolnych odpowiadających T_1, T_2 itd. Na końcu cyklu rozkazu jednostka sterująca musi przekazać sygnał zwrotny do licznika w celu zainicjowania go w jednostce czasu T_1.

Jednostka sterująca zawierająca oba omówione udoskonalenia została przedstawiona na rys. 16.10.

Rozwiązania logiczne jednostki sterującej

Do zdefiniowania sprzętowego rozwiązania jednostki sterującej pozostaje jedynie omówić wewnętrzne układy logiczne tej jednostki, które tworzą wyjściowe sygnały sterujące będące funkcją sygnałów wejściowych.

W tym celu należy obliczyć wyrażenie Boole'a dla każdego sygnału sterującego jako funkcję sygnałów wejściowych. Najlepiej wyjaśnić to na przykładzie. Rozważmy ponownie nasz prosty przykład zilustrowany na rys. 16.5. W tabeli 16.1 zapoznaliśmy się z sekwencjami mikrooperacji i sygnałami sterującymi potrzebnymi do sterowania trzema spośród czterech faz cyklu rozkazu.

Rozważmy pojedynczy sygnał sterujący C_5. Sygnał ten powoduje wczytanie danych z zewnętrznej magistrali danych do rejestru MBR. Widzimy, że w tabeli 16.1 został on użyty dwukrotnie. Zdefiniujmy dwa nowe sygnały sterujące, P i Q, interpretowane następująco:

```
PQ = 00 cykl pobierania
PQ = 01 cykl adresowania pośredniego
PQ = 10 cykl wykonywania
PQ = 11 cykl przerwania
```

Wobec tego następujące wyrażenie Boole'a definiuje C_5:

$$C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2$$

Oznacza to, że sygnał C_5 będzie potwierdzony podczas drugiej jednostki czasowej cyklu pobierania i cyklu adresowania pośredniego.

Wyrażenie to nie jest kompletne. Sygnał C_5 jest również potrzebny podczas cyklu wykonywania. W naszym prostym przykładzie załóżmy, że istnieją tylko trzy rozkazy odczytywane z pamięci: LDA, ADD i AND. Możemy teraz zdefiniować C_5 jako:

$$C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2 + P \cdot \overline{Q} \cdot (\mathrm{LDA} + \mathrm{ADD} + \mathrm{AND}) \cdot T_2$$

Podobny proces mógłby być powtórzony dla każdego sygnału sterującego generowanego przez procesor. Wynikiem byłby zbiór równań Boole'a definiujący zachowanie jednostki sterującej i tym samym procesora.

Aby to wszystko powiązać, jednostka sterująca musi kontrolować stan cyklu rozkazu. Jak już wspomnieliśmy, na zakończenie każdego podcyklu (pobierania, adresowania pośredniego, wykonywania, przerwania) jednostka sterująca wysyła sygnał, który powoduje inicjowanie generatora taktowania i podanie sygnału T_1. Jednostka sterująca musi także ustawić odpowiednie wartości P i Q w celu określenia następnego podcyklu.

Zauważmy, że w przypadku nowoczesnego, złożonego procesora liczba równań Boole'a wymagana do zdefiniowania jednostki sterującej jest bardzo duża. Zadanie wdrożenia układu kombinacyjnego spełniającego te wszystkie równania staje się ekstremalnie trudne. W rezultacie używa się zwykle daleko prostszego rozwiązania, znanego jako *mikroprogramowanie*. Jest ono przedmiotem następnego rozdziału.

16.4. Polecana literatura

W wielu książkach zawarto podstawy działania jednostki sterującej; dwa szczególnie klarowne ujęcia to [HAYE98] i [MANO01].

HAYE98 Hayes J.: *Computer Architecture and Organization*. New York, McGraw-Hill, 1998.
MANO01 Mano M.: *Logic and Computer Design Fundamentals*. Upper Saddle River, Prentice Hall, 2001.

16.5. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Implementacja układowa – *hardwired implementation*
Jednostka sterująca – *control unit*
Magistrala sterowania – *control bus*
Mikrooperacje – *microoperations*
Sygnał sterowania – *control signal*
Ścieżka sterowania – *control path*

Pytania kontrolne

16.1. Objaśnij rozróżnienie między napisaną a czasową sekwencją rozkazu.
16.2. Jaki jest związek między rozkazami a mikrooperacjami?
16.3. Jaka jest ogólna funkcja jednostki sterującej procesora?
16.4. Naszkicuj trzyetapowy proces prowadzący do scharakteryzowania jednostki sterującej.
16.5. Jakie podstawowe zadania realizuje jednostka sterująca?
16.6. Przedstaw wykaz typowych sygnałów wejściowych i wyjściowych jednostki sterującej.
16.7. Wymień trzy rodzaje sygnałów sterujących.
16.8. Krótko objaśnij, co się rozumie przez układową implementację jednostki sterującej.

Problemy do rozwiązania

16.1. Twoja ALU jest w stanie dodawać zawartość swoich dwóch rejestrów wejściowych i może tworzyć logiczne uzupełnienia bitów każdego z rejestrów, nie może jednak odejmować. Liczby są przechowywane w postaci reprezentacji uzupełnienia do dwóch. Wymień mikrooperacje, jakie musi spowodować jednostka sterująca w celu wykonania odejmowania.

16.2. Pokaż mikrooperacje i sygnały sterujące w ujęciu takim, jak w tabeli 16.1 dla CPU z rys. 16.5 i dla następujących rozkazów:

- ładuj akumulator;
- zapisz zawartość akumulatora;
- dodaj do akumulatora;

- AND do akumulatora;
- skocz;
- skocz, jeśli AC = 0;
- dopełnij akumulator.

16.3. Załóż, że opóźnienie propagacji w magistrali i w ALU przedstawionej na rys. 16.6 wynosi odpowiednio 20 i 100 ns. Czas wymagany do skopiowania przez rejestr danych z magistrali wynosi 10 ns. Ile czasu musi być przewidziane na:

(a) przeniesienie danych z rejestru do rejestru?
(b) inkrementowanie licznika programu?

16.4. Napisz sekwencję mikrooperacji wymaganą w przypadku struktury magistralowej z rys. 16.6 dla dodania liczby do AC, jeśli liczba ta jest:

(a) argumentem natychmiastowym;
(b) argumentem o adresie pośrednim;
(c) argumentem o adresie bezpośrednim.

16.5. Wdrożony jest stos w postaci przedstawionej na rys. 10.14. Pokaż sekwencję mikrooperacji dla operacji:

(a) zdejmowania ze stosu;
(b) umieszczania na stosie.

Rozdział 17

Sterowanie mikroprogramowe

17.1. Koncepcje podstawowe
Mikrorozkazy
Mikroprogramowana jednostka sterująca
Sterowanie według Wilkesa
Zalety i wady
17.2. Szeregowanie mikrorozkazów
Rozważania projektowe
Metody szeregowania
Generowanie adresu
Szeregowanie mikrorozkazów w LSI-11
17.3. Wykonywanie mikrorozkazów
Taksonomia mikrorozkazów
Kodowanie mikrorozkazów
Wykonywanie mikrorozkazów w LSI-11
Wykonywanie mikrorozkazów w IBM 3033
17.4. Moduł TI 8800
Format mikrorozkazu
Mikrosekwenser
Rejestrowa ALU
17.5. Zastosowania mikroprogramowania
17.6. Polecana literatura
17.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

Podstawowe spostrzeżenia

- Rozwiązaniem alternatywnym w stosunku do układowej jednostki sterującej jest mikroprogramowana jednostka sterująca, w której logika działania jednostki jest określana za pomocą mikroprogramu. Mikroprogram składa się z sekwencji rozkazów w języku mikroprogramowania. Są to rozkazy bardzo proste, określające mikrooperacje.
- Mikroprogramowana jednostka sterująca jest stosunkowo prostym układem logicznym, zdolnym do (1) szeregowania mikrorozkazów i (2) generowania sygnałów sterujących w celu wykonania tych mikrorozkazów.
- Podobnie jak w układowej jednostce sterującej, sygnały sterowania generowane przez mikrorozkazy są używane do uruchamiania transferów rejestrowych i operacji ALU.

Termin *mikroprogram* został po raz pierwszy zaproponowany przez M. V. Wilkesa we wczesnych latach pięćdziesiątych. Wilkes zaproponował pewien rodzaj uporządkowanego i systematycznego podejścia do projektowania jednostki sterującej, w którym unika się złożoności charakterystycznej dla wdrożenia układowego. Idea ta zaintrygowała wielu badaczy, jednak wydawała się niemożliwa do zrealizowania, ponieważ wymagałaby szybkiej i stosunkowo taniej pamięci sterującej.

Stan sztuki mikroprogramowania przedstawiono w lutowym wydaniu *Datamation* z roku 1964. Żaden system mikroprogramowany nie był w tym czasie szeroko stosowany, a w jednym z artykułów [HILL64] następująco podsumowano popularny pogląd na przyszłość mikroprogramowania: „Jest ona nieco mglista. Żaden z dużych producentów nie wykazał zainteresowania tą metodą, chociaż najprawdopodobniej wszyscy ją zbadali".

Sytuacja ta zmieniła się drastycznie w ciągu niewielu miesięcy. W kwietniu zaanonsowano System/360 IBM, w którym prawie wszystkie największe modele były mikroprogramowane. Chociaż seria 360 wyprzedziła dostępność półprzewodnikowych pamięci ROM, zalety mikroprogramowania okazały się na tyle istotne, że firma IBM zdecydowała się na ten krok. Od tego czasu popularność mikroprogramowania w wielu zastosowaniach wzrastała, przy czym jednym z nich było mikroprogramowe rozwiązanie jednostki sterującej procesora. To właśnie zastosowanie przeanalizujemy w tym rozdziale.

17.1. Koncepcje podstawowe

Mikrorozkazy

W ujęciu dotychczas opisanym jednostka sterująca wydaje się być urządzeniem rozsądnie prostym. Jednak realizacja jednostki sterującej w postaci połączenia podstawowych elementów logicznych jest zadaniem niełatwym. Projekt musi

uwzględniać rozwiązania logiczne służące do porządkowania mikrooperacji, do ich wykonywania, do interpretowania kodów operacji i do podejmowania decyzji na podstawie znaczników stanu ALU. Trudno jest zaprojektować i przetestować takie urządzenie. Ponadto rozwiązanie takie jest stosunkowo nieelastyczne. Trudno jest na przykład zmienić projekt, jeśli ktoś życzy sobie dodania nowego rozkazu maszynowego.

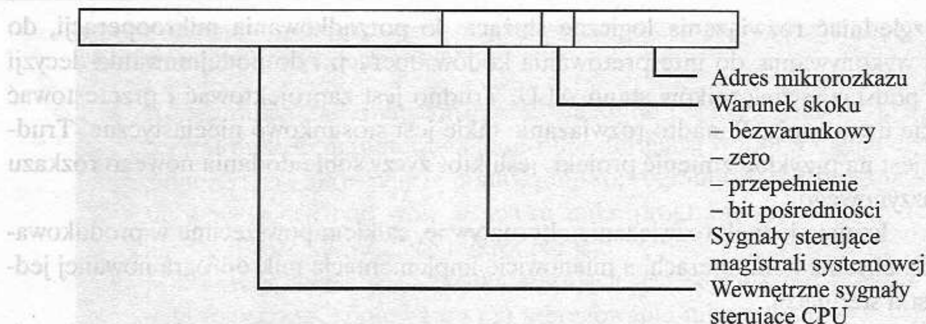
Istnieje jednak rozwiązanie alternatywne, całkiem powszechne w produkowanych obecnie komputerach, a mianowicie implementacja mikroprogramowanej jednostki sterującej.

Rozważmy ponownie tabelę 16.1. Obok wymienionych sygnałów sterujących każda mikrooperacja została opisana w notacji symbolicznej. Zauważmy, że notacja ta wygląda całkiem jak język programowania! W istocie jest to język i nazywa się go *językiem mikroprogramowania*. Każdy wiersz opisuje zbiór jednocześnie następujących mikrooperacji i jest określany jako **mikrorozkaz**. Sekwencja rozkazów jest nazywana *mikroprogramem* lub *oprogramowaniem układowym* (*firmware*). Ten ostatni termin odzwierciedla to, że mikroprogram znajduje się pośrodku między sprzętem a oprogramowaniem. Łatwiej jest zaprojektować oprogramowanie układowe niż sprzęt, jednak trudniej jest napisać program należący do oprogramowania układowego niż do zwykłego oprogramowania.

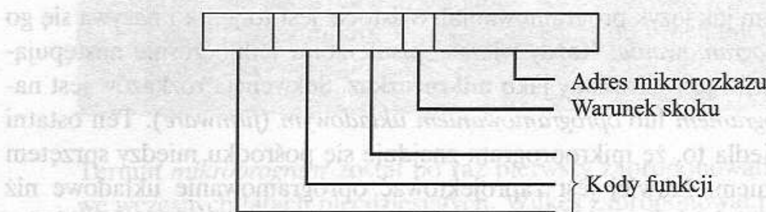
Jak można zastosować koncepcję mikroprogramowania do wdrożenia jednostki sterującej? Zauważmy, że dla każdej mikrooperacji wszystkim, co wykonuje jednostka sterująca, jest generowanie zbioru sygnałów sterujących. Wobec tego dla dowolnej mikrooperacji każda linia sterowania wyprowadzona z jednostki sterującej jest albo w stanie włączenia, albo wyłączenia. Stan ten może być oczywiście reprezentowany przez cyfrę binarną związaną z każdą linią sterowania. Moglibyśmy więc utworzyć *słowo sterujące*, w którym każdy bit reprezentuje jedną linię sterowania. Wobec tego każda mikrooperacja byłaby reprezentowana przez różny układ jedynek i zer w słowie sterującym.

Załóżmy, że połączymy w łańcuch sekwencję słów sterujących w celu reprezentowania sekwencji mikrooperacji wykonywanych przez jednostkę sterującą. Musimy następnie uprzytomnić sobie, że sekwencja mikrooperacji nie jest ustalona. Czasem występuje cykl adresowania pośredniego, a czasem nie. Zapiszmy więc nasze słowa sterujące w pamięci, przy czym każde słowo niech ma unikatowy adres. Dodajmy teraz pole adresowe do każdego słowa sterującego, wskazując w ten sposób lokację następnego słowa przewidzianego do wykonania, jeśli będzie spełniony pewien warunek (np. gdy bit adresowania pośredniego w rozkazie odnoszącym się do pamięci jest równy 1). Dodajmy również kilka bitów precyzujących warunek.

Rezultat jest określany jako *mikrorozkaz poziomy* (rys. 17.1a). Format mikrorozkazu lub słowa sterującego jest następujący. Występuje w nim po jednym bicie na każdą wewnętrzną linię sterowania procesora i po jednym bicie na każdą linię sterowania magistrali systemowej. Występuje też pole warunku wskazujące warunek rozgałęzienia oraz pole adresu następnego mikrorozkazu, który ma być wykonywany, jeśli nastąpi rozgałęzienie.



(a) Mikrorozkaz poziomy



(b) Mikrorozkaz pionowy

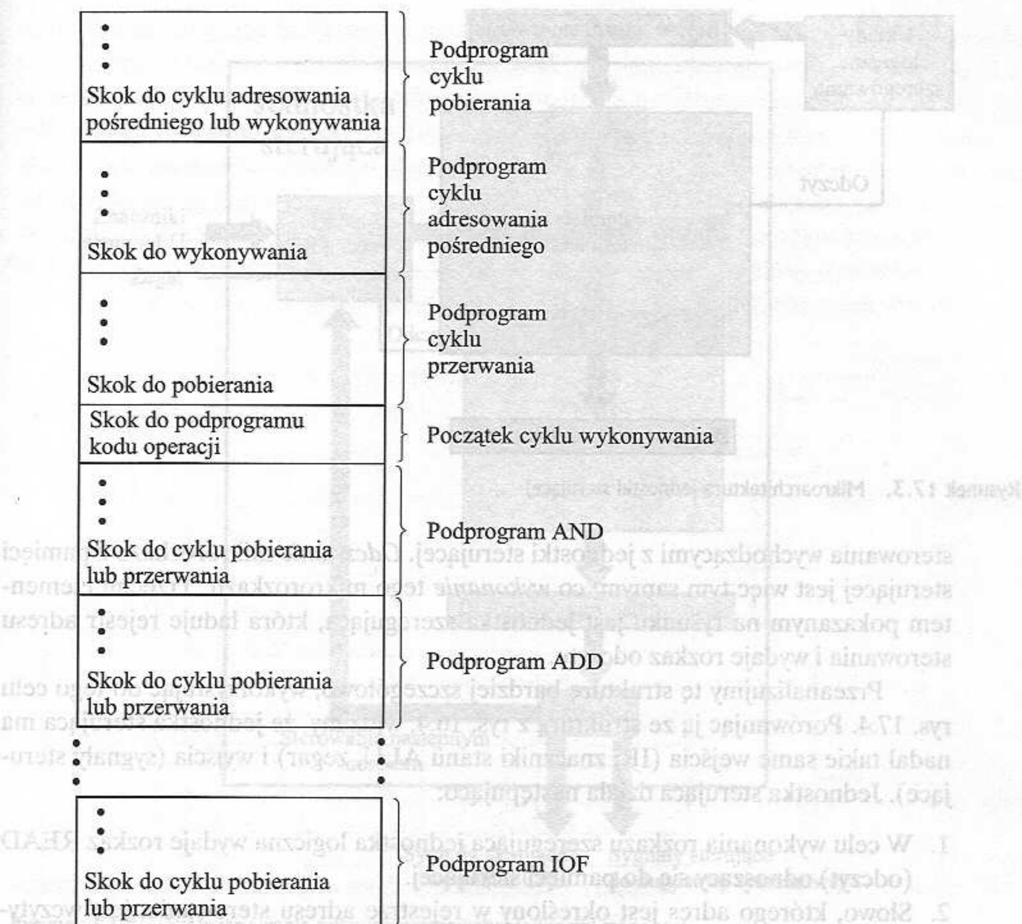
Rysunek 17.1. Typowe formaty mikrorozkazów

Taki mikrorozkaz jest interpretowany następująco:

1. W celu wykonania tego mikrorozkazu należy włączyć wszystkie linie sterowania wskazane przez bity 1 i pozostawić wyłączone wszystkie linie sterowania wskazane przez 0. Wynikające stąd sygnały sterujące spowodują wykonanie jednej lub wielu mikrooperacji.
2. Jeśli warunek wskazany przez bity warunku nie jest spełniony, to należy wykonać następny mikrorozkaz w sekwencji.
3. Jeśli warunek wskazany przez bity warunku jest spełniony, to następny mikrorozkaz przewidziany do wykonania jest wskazany w polu adresowym.

Na rysunku 17.2 widać, w jaki sposób te słowa sterujące lub mikrorozkazy mogą być zorganizowane w postaci *pamięci sterującej*. Mikrorozkazy w każdym programie standardowym mają być realizowane po kolei. Każdy program standardowy kończy się rozkazem rozgałęzienia lub skoku wskazującym, do jakiego punktu należy przejść. Istnieje specjalny program standardowy cyklu wykonywania, którego jedynym celem jest zaznaczenie tego programu związanego z rozkazem maszynowym (AND, ADD itd.), który ma być wykonywany jako następny, zależnie od kodu operacji.

Na rysunku 17.2 jest przedstawiona pamięć sterująca. Rysunek ten jest jednocześnie zwięzłym opisem funkcjonowania jednostki sterującej. W pamięci jest określana sekwencja mikrooperacji, która ma być wykonywana podczas każdego cyklu (pobierania, adresowania pośredniego, wykonywania, przerwania), jest także okreś-



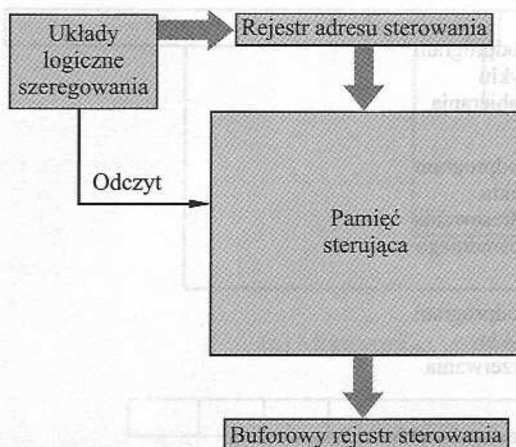
Rysunek 17.2. Organizacja pamięci sterującej

lony porządek tych cykli. Jeśli pamięć ta nie byłaby niczym więcej, ta notacja byłaby pożyteczna do dokumentowania działania jednostki sterującej określonego komputera. Jest ona jednak czymś więcej. Jest także sposobem wdrażania jednostki sterującej.

Mikroprogramowana jednostka sterująca

Pamięć sterująca pokazana na rys. 17.2 zawiera program opisujący zachowanie jednostki sterującej. Wynika stąd, że moglibyśmy wdrożyć jednostkę sterującą, po prostu wykonując ten program.

Na rysunku 17.3 są pokazane podstawowe elementy takiej implementacji. Zbiór mikrorozkazów jest przechowywany w pamięci sterującej. *Rejestr adresu sterowania* zawiera adres następnego mikrorozkazu, który ma być odczytany. Po odczytaniu mikrorozkazu z pamięci sterującej jest on przenoszony do *buforowego rejestru sterowania*. Lewa część tego rejestru (patrz rys. 17.1a) jest połączona z liniami



Rysunek 17.3. Mikroarchitektura jednostki sterującej

sterowania wychodzącymi z jednostki sterującej. *Odczytanie* mikrorozkazu z pamięci sterującej jest więc tym samym, co *wykonanie* tego mikrorozkazu. Trzecim elementem pokazanym na rysunku jest jednostka szeregująca, która ładuje rejestr adresu sterowania i wydaje rozkaz odczytu.

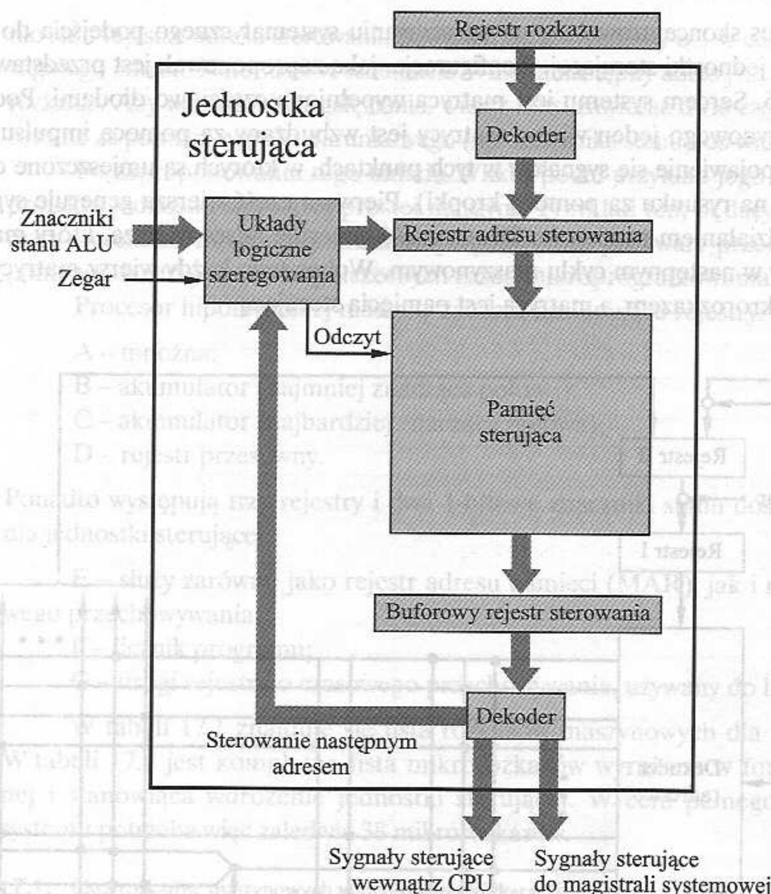
Przeanalizujmy tę strukturę bardziej szczegółowo, wykorzystując do tego celu rys. 17.4. Porównując ją ze strukturą z rys. 16.4, widzimy, że jednostka sterująca ma nadal takie same wejścia (IR, znaczniki stanu ALU, zegar) i wyjścia (sygnały sterujące). Jednostka sterująca działa następująco:

1. W celu wykonania rozkazu szeregująca jednostka logiczna wydaje rozkaz READ (odczyt) odnoszący się do pamięci sterującej.
2. Słowo, którego adres jest określony w rejestrze adresu sterowania, jest wczytywane do buforowego rejestru sterowania.
3. Zawartość buforowego rejestru sterowania generuje sygnały sterujące oraz informację o następnym adresie skierowaną do logicznej jednostki szeregowania.
4. Logiczna jednostka szeregowania ładuje nowy adres do rejestru adresu sterowania na podstawie informacji o następnym adresie uzyskanej z buforowego rejestru sterowania i na podstawie znaczników stanu ALU.

Wszystko to następuje podczas jednego cyklu zegara.

Ostatni z wymienionych kroków wymaga szczegółowego zbadania. Na zakończenie każdego mikrorozkazu logiczna jednostka szeregowania ładuje nowy adres do rejestru adresu sterowania. Zależnie od wartości znaczników stanu ALU i buforowego rejestru sterowania jest podejmowana jedna z trzech decyzji:

- **Przejście do następnego rozkazub** Dodanie 1 do rejestru adresu sterowania.
- **Skok do następnego programu standardowego w wyniku mikrorozkazu skoku.** Ładowanie pola adresu buforowego rejestru sterowania do rejestru adresu sterowania.
- **Skok do programu standardowego kodu maszynowego.** Ładowanie rejestru adresu sterowania na podstawie kodu operacji w IR.



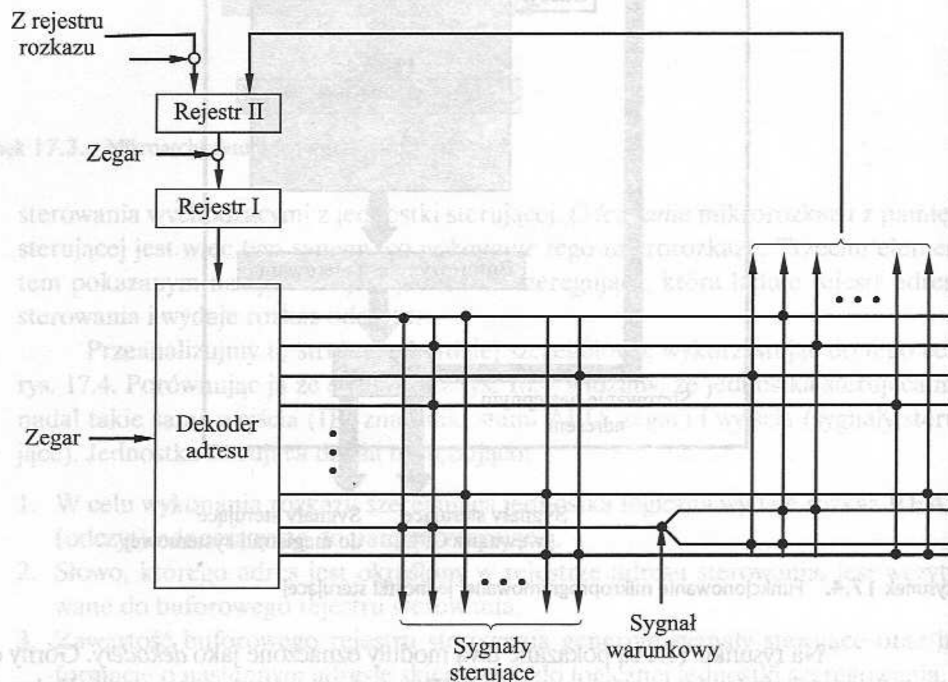
Rysunek 17.4. Funkcjonowanie mikroprogramowanej jednostki sterującej

Na rysunku 17.4 są pokazane dwa moduły oznaczone jako *dekodery*. Górny dekoder przekształca kod operacji z rejestru IR na adres w pamięci sterującej. Dolny dekoder nie jest używany w przypadku mikrorozkazów poziomych, jest natomiast potrzebny w przypadku mikrorozkazów pionowych (rys. 17.1b). Jak już wspomnieliśmy, w mikrorozkazie poziomym każdy bit pola sterowania jest związany z linią sterowania. W mikrorozkazie pionowym dla każdego działania (np. MAR ← (PC)) jest używany kod, a dekoder przekształca ten kod na indywidualne sygnały sterujące. Zaletą mikrorozkazów pionowych jest to, że są one bardziej zwarte (zawierają mniej bitów) niż mikrorozkazy poziome, kosztem niewielkiej ilości dodatkowych układów logicznych oraz opóźnienia.

Sterowanie według Wilkesa

Jak już wspomnieliśmy, Wilkes jako pierwszy zaproponował w roku 1951 użycie mikroprogramowanej jednostki sterującej [WILK51]. Propozycja ta została następnie opracowana w postaci bardziej szczegółowego projektu [WILK53]. Pouczające jest przeanalizowanie tej źródłowej propozycji.

Wilkes skoncentrował się na opracowaniu systematycznego podejścia do projektowania jednostki sterującej. Konfiguracja, jaką zaproponował, jest przedstawiona na rys. 17.5. Sercem systemu jest matryca wypełniona częściowo diodami. Podczas cyklu maszynowego jeden wiersz matrycy jest wzbudzany za pomocą impulsu. Powoduje to pojawienie się sygnałów w tych punktach, w których są umieszczone diody (wskazane na rysunku za pomocą kropki). Pierwsza część wiersza generuje sygnały sterujące działaniem procesora. Druga część generuje adres wiersza, który ma być wzbudzany w następnym cyklu maszynowym. Wobec tego każdy wiersz matrycy jest jednym mikrorozkazem, a matryca jest pamięcią sterującą.



Rysunek 17.5. Mikroprogramowana jednostka sterująca Wilkesa

Na początku cyklu adres wiersza, który ma być pobudzany, znajduje się w rejestrze I. Adres ten jest wprowadzany do dekodera, który, jeśli jest aktywowany przez impuls zegara, pobudza jeden wiersz matrycy. Zależnie od sygnałów sterowania albo kod operacji pochodzący z rejestru rozkazu, albo druga część wzbudzonego wiersza są podczas tego cyklu kierowane do rejestru II. Zawartość rejestru II jest następnie bramkowana do rejestru I za pomocą impulsu zegara. Kolejne impulsy zegara są używane do wzbudzania wiersza matrycy i do przenoszenia zawartości rejestru II do rejestru I. Układ dwurejestrowy jest konieczny, ponieważ dekoder jest po prostu układem kombinacyjnym. Przy zastosowaniu tylko jednego rejestru, wartość wejściowa stałaby się wartością wyjściową w ciągu tego samego cyklu, co spowodowałoby niestabilność.

Schemat ten jest bardzo podobny do mikroprogramowania poziomego opisanego wcześniej (rys. 17.1a). Główna różnica jest następująca. W układzie opisanym poprzed-

nio stan rejestru adresu sterowania powinien być zwiększony o 1 w celu otrzymania następnego adresu. Natomiast w schemacie Wilkesa następny adres jest zawarty w mikrorozkazie. Aby umożliwić rozgałęziania, wiersz musi zawierać dwie części adresowe sterowane za pomocą sygnału warunkowego (np. znacznika stanu), co widać na rysunku.

Po zaproponowaniu tego układu Wilkes podał przykład jego użycia w postaci projektu jednostki sterującej prostej maszyny. Przykład ten, będący pierwszym znanym projektem mikroprogramowanego procesora, jest wart przedstawienia tutaj, ponieważ ilustruje wiele współczesnych zasad mikroprogramowania.

Procesor hipotetycznej maszyny zawiera następujące rejestry:

A – mnożna;
B – akumulator (najmniej znacząca połowa);
C – akumulator (najbardziej znacząca połowa);
D – rejestr przesuwny.

Ponadto występują trzy rejestry i dwa 1-bitowe znaczniki stanu dostępne wyłącznie dla jednostki sterującej:

E – służy zarówno jako rejestr adresu pamięci (MAR), jak i rejestr do czasowego przechowywania;
F – licznik programu;
G – drugi rejestr do czasowego przechowywania, używany do liczenia.

W tabeli 17.1 znajduje się lista rozkazów maszynowych dla tego przykładu. W tabeli 17.2 jest kompletna lista mikrorozkazów wyrażona w formie symbolicznej i stanowiąca wdrożenie jednostki sterującej. W celu pełnego zdefiniowania systemu potrzeba więc zaledwie 38 mikrorozkazów.

Tabela 17.1. Lista rozkazów maszynowych w przykładzie Wilkesa

Rozkaz	Efekt rozkazu
$A\,n$	$C(Acc) + C(n)$ do Acc_1
$S\,n$	$C(Acc) - C(n)$ do Acc_1
$H\,n$	$C(n)$ do Acc_2
$V\,n$	$C(Acc_2) \times C(n)$ do Acc, gdzie $C(n) \geqslant 0$
$T\,n$	$C(Acc_1)$ do n, 0 do Acc
$U\,n$	$C(Acc_1)$ do n
$R\,n$	$C(Acc) \times 2^{-(n+1)}$ do Acc
$L\,n$	$C(Acc) \times 2^{n+1}$ do Acc
$G\,n$	Jeśli $C(Acc) < 0$, to przenieś sterowanie do n; jeśli $C(Acc) \geqslant 0$, to ignoruj (tzn. kontynuuj szeregowo)
$I\,n$	Wczytaj następny znak na wejściu do n
$O\,n$	Wyślij $C(n)$ do wyjścia

Notacja: Acc – akumulator
Acc_1 – najbardziej znacząca połowa bitów akumulatora
Acc_2 – najmniej znacząca połowa bitów akumulatora
n – lokacja pamięci n
$C(X)$ – zawartość X (X – rejestr lub lokacja pamięci)

Tabela 17.2. Mikrorozkazy w przykładzie Wilkesa

Notacja: *A*, *B*, *C*, ... reprezentują różne rejestry w jednostce arytmetycznej i w jednostce sterującej. ´*C* do *D*´ wskazuje, że układy przełączające łączą wyjście rejestru *C* z rejestrem wejściowym *D*; ´(*D* + *A*) do *C*´ wskazuje, że rejestr wyjściowy *A* jest połączony z jednym z wejść sumatora (wyjście *D* jest na stałe dołączone do innego wejścia), a wyjście sumatora z rejestrem *C*. Symbol numeryczny *n* w cudzysłowie (np. ´*n*´) reprezentuje źródło, którego wyjściem jest liczba *n* wyrażona w jednostkach najmniej znaczącej cyfry.

	Jednostka arytmetyczna	Rejestrowa jednostka sterująca	Przerzutnik warunkowy		Następny mikrorozkaz	
			Ustawienie	Użycie	0	1
0		*F* do *G* i *E*			1	
1		(*G* do ´1´) do *F*			2	
2		Pamięć do *G*			3	
3		*G* do *E*			4	
4		*E* do dekodera			–	
A 5	*C* do *D*				16	
S 6	*C* do *D*				17	
H 7	Pamięć do *B*				0	
V 8	Pamięć do *A*				27	
T 9	*C* do pamięci				25	
U 10	*C* do pamięci				0	
R 11	*B* do *D*	*E* do *G*			19	
L 12	*C* do *D*	*E* do *G*			22	
G 13		*E* do *G*	(1)C_5		18	
I 14	Wejście do pamięci				0	
O 15	Pamięć do wyjścia				0	
16	(*D* + pamięć) do *C*				0	
17	(*D* – pamięć) do *C*				0	
18				1	0	1
19	*D* do *B* (*R*)*	(*G* – ´1´) do *E*			20	
20	*C* do *D*		(1)E_5		21	
21	*D* do *C* (*R*)			1	11	0
22	*D* do *C* (*L*)**	(*G* – ´1´) do *E*			23	
23	*B* do *D*		(1)E_5		24	
24	*D* do *B* (*L*)			1	12	0
25	´0´ do *B*				26	
26	*B* do *C*				0	
27	´0´ do *C*	´18´ do *E*			28	
28	*B* do *D*	*E* do *G*	(1)B_1		29	
29	*D* do *B* (*R*)	(*G* – ´1´) do *E*			30	
30	*C* do *D* (*R*)		(2)E_5	1	31	32
31	*D* do *C*			2	28	33
32	(*D* + *A*) do *C*			2	28	33
33	*B* do *D*		(1)B_1		34	
34	*D* do *B* (*R*)				35	
35	*C* do *D* (*R*)			1	36	37
36	*D* do *C*				0	
37	(*D* – *A*) do *C*				0	

*Przesunięcie w prawo. Układy przełączające w jednostce arytmetycznej są połączone tak, że najmniej znacząca cyfra rejestru *C* jest umieszczana podczas mikrooperacji przesuwania w prawo w najbardziej znaczącym miejscu rejestru *B*, a najbardziej znacząca cyfra rejestru *C* (cyfra znaku) jest powtarzana (dzięki temu jest wykonywana poprawka w przypadku liczb ujemnych).

**Przesunięcie w lewo. Układy przełączające są tak połączone, aby podczas mikrooperacji przesunięcia w lewo przepuścić najbardziej znaczącą cyfrę rejestru *B* do najmniej znaczącego miejsca rejestru *C*.

W pierwszej kolumnie jest podany adres (numer wiersza) każdego mikrorozkazu. Adresy te odpowiadają wymienionym kodom operacji. Gdy więc jest napotkany kod operacji rozkazu dodawania (A), to wykonywany jest mikrorozkaz zawarty w lokacji 5. W kolumnach 2 i 3 są przedstawione działania realizowane odpowiednio przez ALU i jednostkę sterującą. Każde wyrażenie symboliczne musi być przekształcone na zbiór sygnałów sterujących (bitów mikrorozkazu). Kolumny 4 i 5 dotyczą ustawiania i używania dwóch znaczników stanu (przerzutników). W kolumnie 4 jest określony sygnał, który powoduje ustawienie znaczników stanu. Na przykład $(1)C_s$ oznacza, że znacznik stanu numer 1 jest ustawiany przez bit znaku liczby w rejestrze C. Jeśli kolumna 5 zawiera identyfikator znacznika stanu, to kolumny 6 i 7 zawierają dwa alternatywne adresy mikrorozkazów, które mają być użyte. W przeciwnym razie kolumna 6 określa adres następnego mikrorozkazu do pobrania.

Rozkazy od 0 do 4 składają się na cykl pobierania. Mikrorozkaz 4 przekazuje kod operacji do dekodera, który generuje adres mikrorozkazu odpowiadający przewidzianemu do pobrania rozkazowi maszynowemu. Czytelnik, na podstawie dokładnego przestudiowania tabeli 17.2, powinien być w stanie zrozumieć działanie jednostki sterującej.

Zalety i wady

Główną zaletą użycia mikroprogramowania do implementacji jednostki sterującej jest uproszczenie projektowania tej jednostki. Rozwiązanie takie jest zarówno tańsze, jak i bardziej odporne na błędy. *Układowa* jednostka sterująca musi zawierać skomplikowane układy logiczne w celu szeregowania wielu mikrooperacji w cyklu rozkazu. Jednak dekodery i logiczna jednostka szeregowania w mikroprogramowanej jednostce sterującej są bardzo prostymi układami logicznymi.

Główną wadą mikroprogramowanej jednostki sterującej jest to, że jest ona nieco wolniejsza od jednostki układowej wykonanej w porównywalnej technologii. Mimo to mikroprogramowanie jest dominującą metodą wdrażania jednostek sterujących we współczesnych komputerach, przede wszystkim z powodu łatwości stosowania. W procesorach RISC, charakteryzujących się prostszym formatem rozkazu, są stosowane zwykle układowe jednostki sterujące. Teraz przeanalizujemy rozwiązanie mikroprogramowane bardziej szczegółowo.

17.2. Szeregowanie mikrorozkazów

Dwoma podstawowymi zadaniami realizowanymi przez mikroprogramowaną jednostkę sterującą są:

- **Szeregowanie mikrorozkazów.** Odczytywanie następnego mikrorozkazu z pamięci sterującej.
- **Wykonywanie mikrorozkazu.** Generowanie sygnałów sterujących potrzebnych do wykonania mikrorozkazu.

Podczas projektowania jednostki sterującej obydwa te zadania muszą być rozpatrywane łącznie, ponieważ oba mają wpływ na format mikrorozkazu i taktowanie jednostki sterującej. W tym podrozdziale skupimy się na szeregowaniu, w minimalnym stopniu zajmując się zagadnieniami formatu i taktowania. Zagadnienia te zostaną przeanalizowane bardziej szczegółowo w następnym podrozdziale.

Rozważania projektowe

Dwa następujące problemy występują przy projektowaniu metody szeregowania mikrorozkazów: rozmiar mikrorozkazu i czas generowania adresu. Pierwszy problem jest oczywisty; minimalizowanie rozmiaru pamięci sterującej zmniejsza koszt tego zespołu. Drugi problem polega na dążeniu do tak szybkiego wykonywania mikrorozkazów, jak tylko jest to możliwe.

Przy wykonywaniu mikroprogramu adres następnego mikrorozkazu przewidzianego do wykonania należy do jednej z trzech kategorii:

- określany przez rejestr rozkazu;
- następny adres w sekwencji;
- rozgałęzienie.

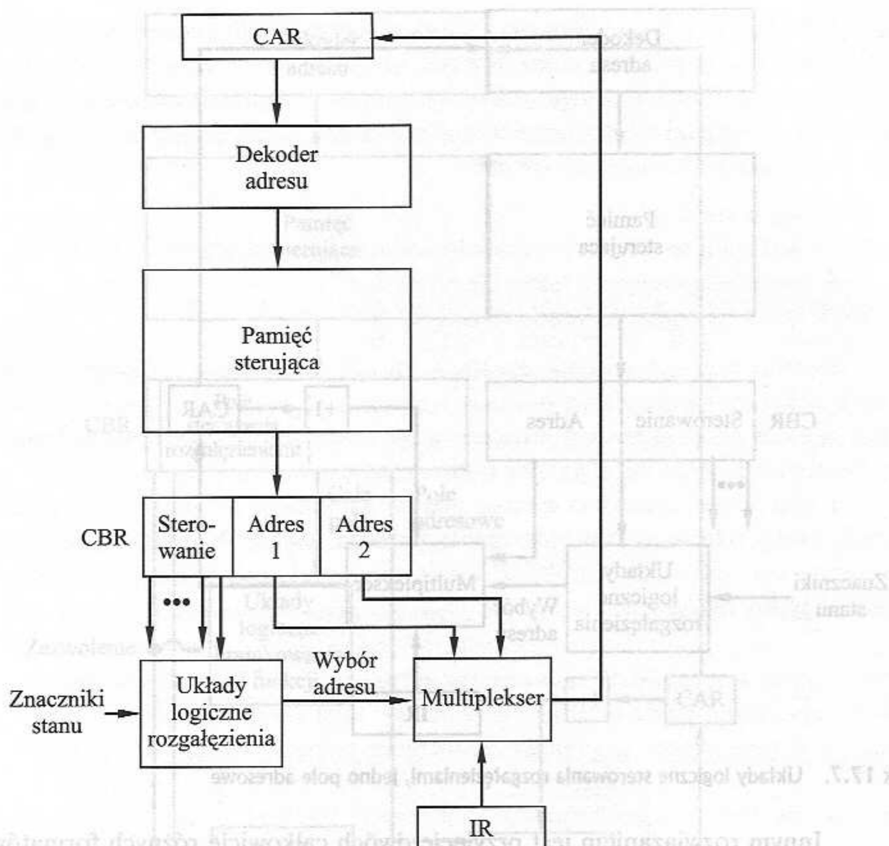
Pierwsza kategoria występuje tylko raz w cyklu rozkazu, tuż po pobraniu rozkazu. Druga kategoria jest najbardziej powszechna w większości projektów. Nie jest jednak możliwa taka optymalizacja projektu, żeby występowały tylko dostępy sekwencyjne. Rozgałęzienia, zarówno warunkowe, jak i bezwarunkowe, są niezbędną częścią mikroprogramu. Ponadto dąży się do używania krótkich sekwencji mikrorozkazów; jeden z każdych trzech lub czterech mikrorozkazów powinien dotyczyć rozgałęzienia [SIEW82]. Ważne jest więc projektowanie zwartych, wydajnych czasowo metod wykonywania mikrorozkazów rozgałęziania.

Metody szeregowania

Adres następnego mikrorozkazu w pamięci sterującej musi być generowany na podstawie bieżącego mikrorozkazu, warunkowych znaczników stanu oraz zawartości rejestru rozkazu. Używa się do tego celu wielu różnych metod. Możemy je pogrupować na trzy ogólne kategorie zilustrowane na rys. 17.6÷17.8. Kategorie te są oparte na formacie informacji adresowej w mikrorozkazie:

- dwa pola adresowe;
- jedno pole adresowe;
- format zmienny.

Najprostszym rozwiązaniem jest uwzględnienie dwóch pól adresowych w każdym mikrorozkazie. Na rysunku 17.6 zasugerowano sposób użycia tej informacji. Występuje tu multiplekser służący jako miejsce docelowe obu pól adresowych i zawartości rejestru rozkazu. Na podstawie wejścia wyboru adresu multiplekser przekazuje do rejestru adresu sterowania (CAR) albo kod operacji, albo jeden z dwóch



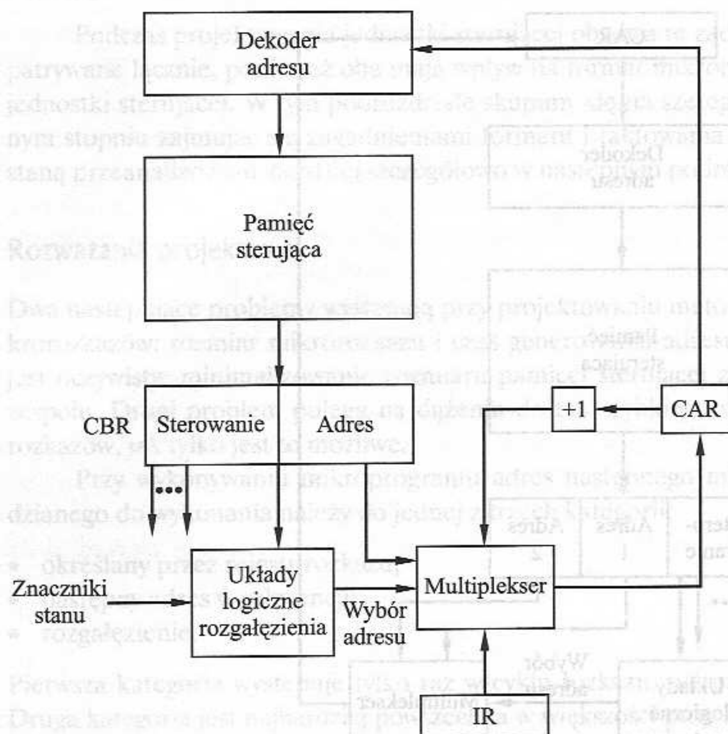
Rysunek 17.6. Układy logiczne sterowania rozgałęzieniami, dwa pola adresowe

adresów. Zawartość rejestru CAR jest następnie dekodowana w celu utworzenia adresu następnego mikrorozkazu. Sygnały wyboru adresu są doprowadzane przez moduł logiczny rozgałęziania, na którego wejście są podawane znaczniki stanu jednostki sterującej oraz bity pochodzące z części sterowania w mikrorozkazie.

Chociaż rozwiązanie dwuadresowe jest proste, wymaga ono większej ilości bitów w mikrorozkazie niż inne rozwiązania. Oszczędności mogą być poczynione za pomocą pewnych dodatkowych układów logicznych. Powszechnym rozwiązaniem jest stosowanie jednego pola adresowego (rys. 17.7). W tym rozwiązaniu opcje dotyczące pobierania następnego adresu są następujące:

- pole adresowe;
- kod rejestru rozkazu;
- następny adres w sekwencji.

Sygnały wyboru adresu określają, która z tych opcji zostaje wybrana. Rozwiązanie to powoduje zredukowanie liczby pól adresowych do jednego. Zauważmy jednak, że często pole to nie będzie używane. Schemat kodowania mikrorozkazu wykazuje więc pewną nieefektywność.



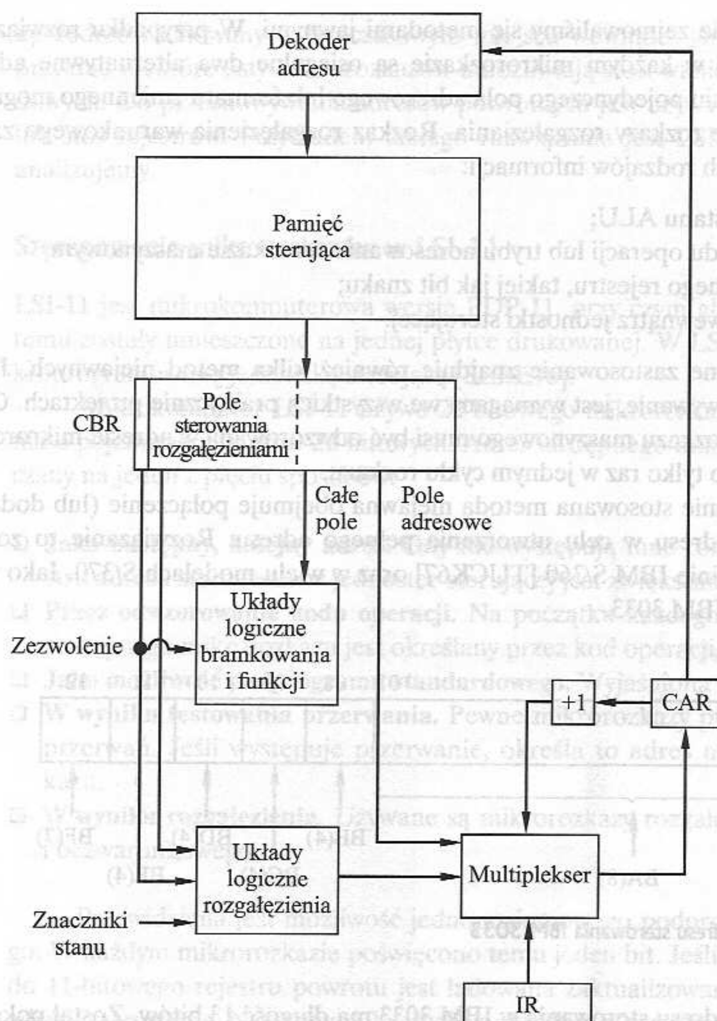
Rysunek 17.7. Układy logiczne sterowania rozgałęzieniami, jedno pole adresowe

Innym rozwiązaniem jest przyjęcie dwóch całkowicie różnych formatów mikrorozkazu (rys. 17.8). Za pomocą jednego bitu określa się, który format będzie używany. W jednym formacie pozostałe bity są używane do aktywowania sygnałów sterujących. W drugim formacie niektóre bity sterują modułem logicznym rozgałęziania, a pozostałe dostarczają adres. W przypadku pierwszego formatu następny adres jest albo następnym adresem w sekwencji, albo adresem wyprowadzanym z rejestru rozkazu. W przypadku drugiego formatu jest określane albo rozgałęzienie warunkowe, albo bezwarunkowe. Wadą tego rozwiązania jest to, że jeden mikrorozkaz rozgałęzienia pochłania cały cykl. W przypadku pozostałych rozwiązań generowanie adresu następuje jako część tego samego cyklu, co generowanie sygnałów sterujących. Powoduje to zminimalizowanie liczby odniesień do pamięci sterującej.

Oczywiście, opisane rozwiązania mają charakter ogólny. Konkretne wdrożenia często obejmują odmiany lub kombinacje tych metod.

Generowanie adresu

Spoglądaliśmy na problem szeregowania pod względem rozważań nad formatem i ogólnych wymagań dotyczących logiki. Innym punktem widzenia jest rozważenie różnych sposobów wyprowadzania lub obliczania następnego adresu.



Rysunek 17.8. Układy logiczne sterowania rozgałęzieniami, format zmienny

W tabeli 17.3 wymieniono różne metody generowania adresu. Mogą one być podzielone na metody jawne, w których adres jest jawnie osiągalny w mikrorozkazie, oraz metody domyślne wymagające dodatkowych układów logicznych do wygenerowania adresu.

Tabela 17.3. Metody generowania adresu mikrorozkazu

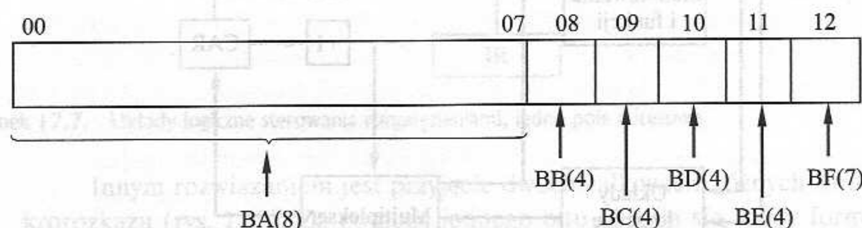
Jawne	Niejawne
Dwupolowa	Odwzorowanie
Rozgałęzienie bezwarunkowe	Dodawanie
Rozgałęzienie warunkowe	Sterowanie szczątkowe

W zasadzie zajmowaliśmy się metodami jawnymi. W przypadku rozwiązania dwuadresowego w każdym mikrorozkazie są osiągalne dwa alternatywne adresy. Przy zastosowaniu pojedynczego pola adresowego lub formatu zmiennego mogą być wdrożone różne rozkazy rozgałęziania. Rozkaz rozgałęzienia warunkowego zależy od następujących rodzajów informacji:

- znaczników stanu ALU;
- części pól kodu operacji lub trybu adresowania w rozkazie maszynowym;
- części wybranego rejestru, takiej jak bit znaku;
- bitów stanu wewnątrz jednostki sterującej.

Powszechne zastosowanie znajduje również kilka metod niejawnych. Jedna z nich, odwzorowywanie, jest wymagana we wszystkich praktycznie projektach. Część kodu operacji rozkazu maszynowego musi być odwzorowana w adresie mikrorozkazu. Następuje to tylko raz w jednym cyklu rozkazu.

Powszechnie stosowana metoda niejawna obejmuje połączenie (lub dodanie) dwóch części adresu w celu utworzenia pełnego adresu. Rozwiązanie to zostało przyjęte w rodzinie IBM S/360 [TUCK67] oraz w wielu modelach S/370. Jako przykładu użyjemy IBM 3033.



Rysunek 17.9. Rejestr adresu sterowania IBM 3033

Rejestr adresu sterowania w IBM 3033 ma długość 13 bitów. Został pokazany na rys. 17.9. Można wyróżnić dwie części adresu. Osiem najstarszych bitów (00-07) normalnie nie ulega zmianie przy przejściu od jednego cyklu mikrorozkazu do następnego. Podczas wykonywania mikrorozkazu tych 8 bitów kopiuje się bezpośrednio z 8-bitowego pola mikrorozkazu (pole BA) do 8 najstarszych bitów rejestru adresu sterowania. W ten sposób definiuje się blok 32 mikrorozkazów w pamięci sterowania. Pozostałych 5 bitów rejestru adresu sterowania jest ustawianych w celu określenia specyficznego adresu mikrorozkazu, który ma być pobrany jako następny. Każdy z tych bitów jest określany przez 4-bitowe pole (z wyjątkiem jednego, który jest związany z polem 7-bitowym) w bieżącym mikrorozkazie; pole określa warunek ustawiania odpowiedniego bitu. Na przykład bit w rejestrze adresu sterowania może być ustawiony jako 1 lub 0 zależnie od tego, czy wystąpiło przeniesienie w ostatniej operacji ALU.

Ostatnie rozwiązanie wymienione w tabeli 17.3 nosi nazwę *sterowania resztkowego* (*residual control*). Obejmuje ono użycie adresu mikrorozkazu, który został

uprzednio zachowany w tymczasowym miejscu wewnątrz jednostki sterującej. Na przykład niektóre listy mikrorozkazów umożliwiają stosowanie programów standardowych. Do przechowywania adresów powrotnych jest używany rejestr wewnętrzny lub stos rejestrów. Przykładem takiego rozwiązania jest LSI-11, który teraz przeanalizujemy.

Szeregowanie mikrorozkazów w LSI-11

LSI-11 jest mikrokomputerową wersją PDP-11, przy czym główne podzespoły systemu zostały umieszczone na jednej płytce drukowanej. W LSI-11 zastosowano mikroprogramowaną jednostkę sterującą [SEBE76].

Mikrokomputer LSI-11 używa 22-bitowego mikrorozkazu i pamięci sterowania o pojemności 2K słów 22-bitowych. Adres następnego mikrorozkazu jest wyznaczany na jeden z pięciu sposobów:

- **Jako następny, kolejny adres.** Gdy nie występują inne rozkazy, zawartość rejestru adresu sterowania w jednostce sterującej jest zwiększana o 1.
- **Przez odwzorowanie kodu operacji.** Na początku każdego cyklu rozkazu adres następnego mikrorozkazu jest określany przez kod operacji.
- **Jako możliwość podprogramu standardowego.** Wyjaśniona poniżej.
- **W wyniku testowania przerwania.** Pewne mikrorozkazy precyzują sprawdzanie przerwań. Jeśli występuje przerwanie, określa to adres następnego mikrorozkazu.
- **W wyniku rozgałęzienia.** Używane są mikrorozkazy rozgałęzienia warunkowego i bezwarunkowego.

Przewidziana jest możliwość jednopoziomowego podprogramu standardowego. W każdym mikrorozkazie poświęcono temu jeden bit. Jeśli jest on ustawiony, to do 11-bitowego rejestru powrotu jest ładowana zaktualizowana zawartość rejestru adresu sterowania. Następny mikrorozkaz określający powrót spowoduje, że rejestr adresu sterowania zostanie załadowany zawartością rejestru powrotu.

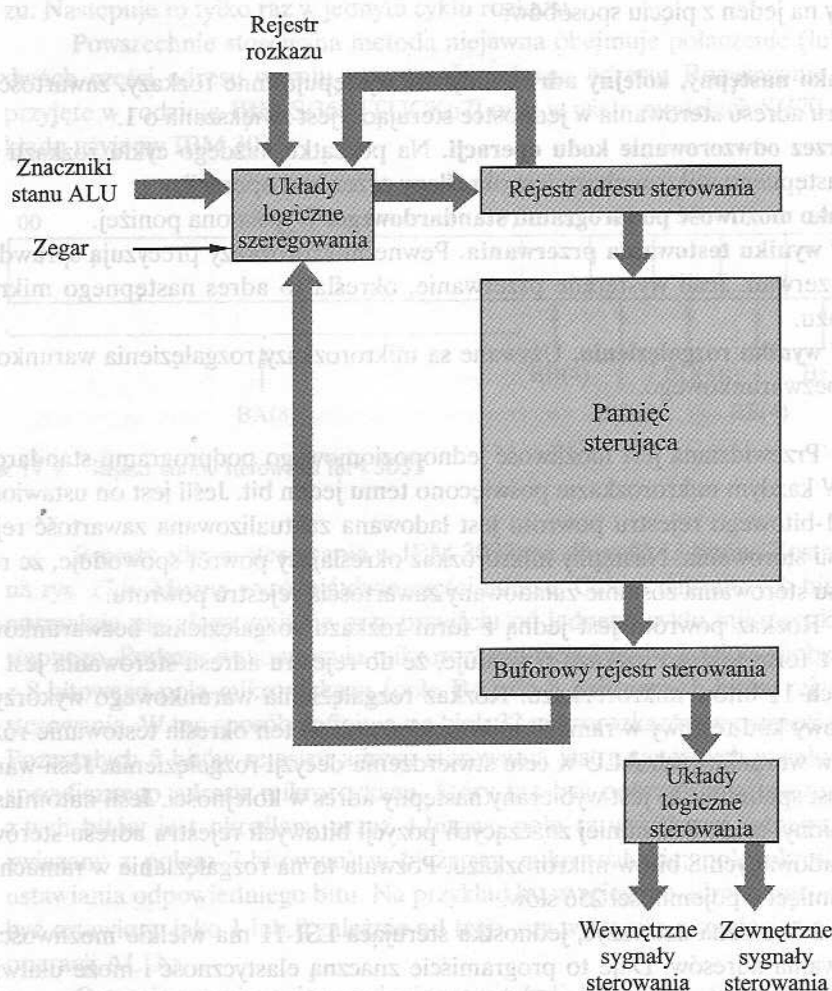
Rozkaz powrotu jest jedną z form rozkazu rozgałęzienia bezwarunkowego. Inna z form takiego rozkazu powoduje, że do rejestru adresu sterowania jest ładowanych 11 bitów mikrorozkazu. Rozkaz rozgałęzienia warunkowego wykorzystuje 4-bitowy kod testowy w ramach mikrorozkazu. Kod ten określa testowanie różnych kodów warunkowych ALU w celu stwierdzenia decyzji rozgałęzienia. Jeśli warunek nie jest spełniony, to jest wybierany następny adres w kolejności. Jeśli natomiast jest spełniony, to do 8 najmniej znaczących pozycji bitowych rejestru adresu sterowania jest ładowanych 8 bitów mikrorozkazu. Pozwala to na rozgałęzianie w ramach strony pamięci o pojemności 256 słów.

Jak można zauważyć, jednostka sterująca LSI-11 ma wielkie możliwości szeregowania adresów. Daje to programiście znaczną elastyczność i może ułatwić zadanie mikroprogramowania. Z drugiej strony takie rozwiązanie wymaga większej liczby układów logicznych w jednostce sterującej niż rozwiązania prostsze.

17.3. Wykonywanie mikrorozkazów

Cykl mikrorozkazu jest podstawowym zdarzeniem w mikroprogramowanym procesorze. Każdy cykl składa się z dwóch części: pobierania i wykonywania. Część pobierania jest określona przez generowanie adresu mikrorozkazu i o tym mówiliśmy w poprzednim podrozdziale. W tym podrozdziale zajmiemy się wykonywaniem mikrorozkazu.

Przypomnijmy sobie, co się dzieje w wyniku wykonania mikrorozkazu. W istocie rezultatem wykonania mikrorozkazu są sygnały sterujące. Niektóre z tych sygnałów sterujących są rozsyłane wewnątrz procesora. Pozostałe są kierowane do zewnętrznej magistrali sterowania lub do zewnętrznego interfejsu. Oprócz tego jest określany adres następnego mikrorozkazu.



Rysunek 17.10. Organizacja jednostki sterującej

Z tego opisu wynika organizacja jednostki sterującej, którą pokazano na rys. 17.10. Ta nieco zrewidowana wersja rys. 17.4 stanowi nasz główny punkt zainteresowania w tym podrozdziale. Rola podstawowych modułów na tym rysunku powinna teraz być oczywista. Logiczny moduł szeregowania zawiera układy logiczne wykonujące funkcje omówione w poprzednim podrozdziale. Generuje on adres następnego mikrorozkazu na podstawie rejestru rozkazu, znaczników stanu ALU, rejestru adresu sterowania (w celu zwiększania jego stanu) oraz buforowego rejestru sterowania. Ten ostatni może dostarczać rzeczywistego adresu, bitów sterowania lub jednego i drugiego. Moduł jest sterowany zegarem, który określa taktowanie cykli mikrorozkazów.

Moduł logiczny sterowania generuje sygnały sterujące jako funkcję niektórych bitów mikrorozkazu. Powinno być jasne, że format i zawartość mikrorozkazu określa złożoność logicznego modułu sterowania.

Taksonomia mikrorozkazów

Mikrorozkazy mogą być klasyfikowane na różne sposoby. W literaturze można znaleźć następujące sposoby podziału mikrorozkazów:

- pionowe – poziome;
- upakowane – nieupakowane;
- mikroprogramowane układowo – programowo;
- kodowane bezpośrednio – pośrednio.

Wszystkie te cechy wpływają na format mikrorozkazu. Żaden z tych terminów nie był używany w literaturze w sposób spójny i precyzyjny. Jednak przeanalizowanie tych rozróżnień jakościowych może posłużyć do wyjaśnienia możliwości występujących przy projektowaniu mikrorozkazów. W następnych punktach rozpatrzymy najpierw istotne problemy projektowania leżące u podstaw kolejnych par właściwości, po czym przedstawimy koncepcje wynikające z każdej pary.

W oryginalnej propozycji Wilkesa [WILK51] każdy bit mikrorozkazu albo bezpośrednio tworzył sygnał sterujący, albo bezpośrednio określał bit następnego adresu. Zobaczyliśmy w poprzednim podrozdziale, że możliwe są bardziej złożone schematy szeregowania adresów, przy użyciu mniejszej liczby bitów mikrorozkazu. Schematy te wymagają bardziej złożonego logicznego modułu szeregującego. Podobny dylemat występuje w odniesieniu do tej części mikrorozkazu, która dotyczy sygnałów sterujących. Można zaoszczędzić bity słowa sterowania przez zakodowanie informacji sterowania, a następnie jej zdekodowanie w celu utworzenia sygnałów sterujących.

Jak można przeprowadzić to kodowanie? Żeby odpowiedzieć na to pytanie, weźmy pod uwagę K różnych, wewnętrznych i zewnętrznych sygnałów sterujących dostarczanych przez jednostkę sterującą. W schemacie Wilkesa do tego celu służy K bitów mikrorozkazu. Pozwala to na generowanie 2^K możliwych kombinacji sygnałów sterujących podczas dowolnego cyklu rozkazu. Możemy to jednak poprawić, zauważając, że nie wszystkie możliwe kombinacje zostaną użyte. Na przykład:

- Dwa źródła nie mogą być bramkowane do tego samego miejsca docelowego (np. C_2 i C_8 na rys. 16.5).
- Rejestr nie może jednocześnie być źródłem i miejscem przeznaczenia (np. C_5 i C_{12} na rys. 16.5).
- W określonej chwili tylko jedna kombinacja sygnałów sterujących może być skierowana do ALU.
- W określonej chwili tylko jedna kombinacja sygnałów sterujących może być doprowadzona do zewnętrznej magistrali sterowania.

Tak więc dla danego procesora można by wymienić wszystkie możliwe i dopuszczalne kombinacje sygnałów sterujących, uzyskując liczbę możliwości $Q < 2^K$. Mogłyby one być zakodowane za pomocą $\log_2 Q$ bitów, przy czym $(\log_2 Q) < K$. Mogłaby to być najbardziej zwarta postać kodowania zachowująca wszystkie dopuszczalne kombinacje sygnałów sterujących. W praktyce ta postać kodowania nie jest używana z dwóch powodów:

- Jest ona równie trudna do programowania, jak czysty schemat zdekodowany (Wilkesa).
- Wymaga ona złożonego i tym samym powolnego modułu logicznego sterowania.

Zamiast tego przyjmuje się pewne rozwiązania kompromisowe. Są one dwóch rodzajów:

- Do zakodowania możliwych kombinacji używa się większej liczby bitów, niż jest ściśle konieczna.
- Niemożliwe jest zakodowanie niektórych fizycznie dopuszczalnych kombinacji.

Ten ostatni rodzaj kompromisu wpływa na zredukowanie liczby bitów. Jednak netto liczba bitów przekracza $\log_2 Q$.

W następnym punkcie przedyskutujemy konkretne metody kodowania. Pozostała część tego punktu jest poświęcona wynikom kodowania oraz różnym wyrażeniom służącym do jego opisu.

Na podstawie dotychczasowych rozważań możemy stwierdzić, że występuje pewne widmo formatów mikrorozkazu w części dotyczącej sygnałów sterujących. Na jednym biegunie mamy do czynienia z jednym bitem na każdy sygnał sterujący, na drugim występuje format w znacznym stopniu zakodowany. W tabeli 17.4 widać, że również i pozostałe właściwości mikroprogramowanych jednostek sterujących tworzą pewne widma i że są one w większości zdeterminowane przez stopień zakodowania.

Druga para właściwości wymieniona w tabeli jest raczej oczywista. Czysty schemat Wilkesa wymaga najwięcej bitów. Powinno być również jasne, że to rozwiązanie reprezentuje najbardziej szczegółowy obraz sprzętu. Każdy sygnał sterujący może być indywidualnie kontrolowany przez programistę. Natomiast kodowanie

jest dokonywane w taki sposób, żeby połączyć funkcje lub zasoby, przez co programista widzi procesor na wyższym, mniej szczegółowym poziomie. Ponadto kodowanie jest projektowane w celu ułatwienia mikroprogramowania. Ponownie zwróćmy uwagę na to, że zadanie zrozumienia i zgrania wszystkich sygnałów sterujących jest zadaniem trudnym. Jak już wspomnieliśmy, jedną z konsekwencji kodowania jest zwykle zapobieżenie używaniu pewnych kombinacji, które w innym przypadku byłyby dopuszczalne.

W poprzednim rozdziale omówiliśmy projektowanie mikrorozkazów z punktu widzenia programisty. Jednak na stopień zakodowania można również spojrzeć pod kątem jego wpływu na sprzęt. W przypadku czystego formatu niezakodowanego potrzeba niewiele lub nie potrzeba wcale dekodujących układów logicznych; każdy bit generuje określony sygnał sterujący. W miarę jak używane są bardziej zwarte i bardziej ogólne schematy kodowania, potrzebne są coraz bardziej złożone układy logiczne dekodowania. To z kolei może niekorzystnie wpływać na wydajność. Na propagację sygnałów przez bramki bardziej złożonego logicznego modułu sterowania potrzeba więcej czasu. Wobec tego wykonywanie zakodowanych mikrorozkazów trwa dłużej niż rozkazów niezakodowanych.

Tabela 17.4. Rodzaje mikrorozkazów

Właściwości	
Niezakodowane	Wysoce zakodowane
Wiele bitów	Kilka bitów
Szczegółowy widok sprzętu	Zagregowany widok sprzętu
Trudne do programowania	Łatwe do programowania
W pełni wykorzystana współbieżność	Nie w pełni wykorzystana współbieżność
Niewiele lub brak sterujących układów logicznych	Złożone sterujące układy logiczne
Szybkie wykonywanie	Powolne wykonywanie
Optymalizacja wydajności	Optymalizacja programowania
Terminologia	
Nieupakowane	Upakowane
Poziome	Pionowe
Układowe	Programowe

Wszystkie właściwości wymienione w tabeli 17.4 tworzą zatem pewne widmo strategii projektowania. Na ogół projektowanie zbliża się do jednego końca widma tego zbioru, w celu zoptymalizowania wydajności jednostki sterującej. Projekty zbliżone do drugiego krańca widma są bardziej ukierunkowane na optymalizację procesu mikroprogramowania. Istotnie, listy mikrorozkazów zbliżone do drugiego krańca widma wyglądają bardzo podobnie do list rozkazów maszynowych. Dobrym tego przykładem jest projekt LSI-11, który opiszemy w tym podrozdziale. Zwykle, gdy celem jest po prostu wdrożenie jednostki sterującej, projekt będzie zbliżony do tego pierwszego krańca zbioru. Projekt IBM 3033, który również omówimy, należy do tej właśnie kategorii. Jak wykażemy w dalszym ciągu, niektóre systemy umożliwiają

różnym użytkownikom budowanie różnych mikroprogramów przy użyciu takich samych rozwiązań mikrorozkazów. W tych przypadkach projekt najprawdopodobniej znajdzie się bliżej drugiego krańca widma.

Możemy teraz posługiwać się wcześniej wprowadzoną terminologią. W tabeli 17.4 jest pokazane, w jaki sposób trzy spośród wymienionych par wyrażeń wiążą się z widmem mikrorozkazów. W istocie, wszystkie te pary opisują ten sam obiekt, jednak akcentują różne cechy projektu.

Stopień upakowania wiąże się ze stopniem identyfikacji między określonym zadaniem sterowania a określonymi bitami mikrorozkazu. Gdy bity stają się bardziej upakowane, dana liczba bitów zawiera więcej informacji. Wobec tego upakowanie wiąże się znaczeniowo z kodowaniem. Terminy *poziomy* i *pionowy* odnoszą się do względnej szerokości mikrorozkazów. W [SIEW82] zasugerowano prostą regułę, zgodnie z którą mikrorozkazy pionowe mają długości w przedziale 16÷40 bitów, poziome zaś w przedziale 40÷100 bitów. Terminy mikroprogramowanie *układowe* i *programowe* są używane w celu zasugerowania stopnia zbliżenia do stanowiących podstawę sygnałów sterujących oraz do rozwiązania sprzętowego. *Mikroprogramy układowe* (*hard microprograms*) są na ogół stałe i osadzone w pamięci stałej. *Mikroprogramy programowe* (*soft microprograms*) można łatwiej zmieniać i przywodzą one na myśl mikroprogramowanie przez użytkownika.

Inna para terminów wymienionych na początku tego podrozdziału odnosi się do kodowania bezpośredniego lub pośredniego. Zajmiemy się teraz tym właśnie rozróżnieniem.

Kodowanie mikrorozkazu

W praktyce nie projektuje się jednostek sterujących jako czysto niekodowanych lub posługujących się poziomym formatem mikrorozkazu. Używa się przynajmniej pewnego stopnia kodowania w celu zredukowania szerokości pamięci sterowania oraz uproszczenia zadania mikroprogramowania.

Podstawowa metoda kodowania jest przedstawiona na rys. 17.11a. Mikrorozkaz jest zorganizowany w postaci zbioru pól. Każde pole zawiera kod, który po zdekodowaniu aktywuje jeden lub więcej sygnałów sterujących.

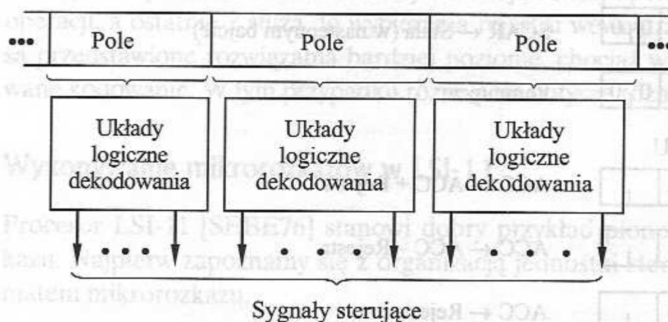
Zastanówmy się, co wynika z tego schematu. Podczas wykonywania mikrorozkazu każde pole jest dekodowane i są generowane sygnały sterujące. Wobec tego przy liczbie N pól jednocześnie prowadzi się N działań. Wynikiem każdego działania jest wysłanie jednego lub wielu sygnałów sterujących. Na ogół, chociaż nie zawsze, chcemy zaprojektować format w ten sposób, żeby każdy sygnał sterujący był wysyłany przez nie więcej niż jedno pole. Oczywiście musi być możliwe, aby każdy sygnał sterujący był aktywowany przez przynajmniej jedno pole.

Rozważmy teraz indywidualne pole. Pole obejmujące L bitów zawiera jeden z 2^L kodów, z których każdy może być przyporządkowany innej kombinacji sygnałów sterujących. Ponieważ w określonej chwili w polu może pojawić się tylko jeden kod, kody wzajemnie wykluczają się i wobec tego wykluczają się wzajemnie również powodowane przez nie działania.

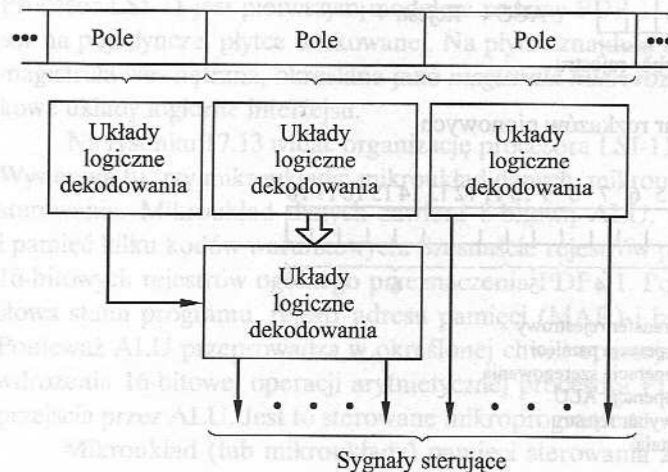
Projektowanie kodowanego formatu mikrorozkazu może więc być wyrażone w prosty sposób:

- Podział formatu na niezależne pola. Oznacza to, że każde pole obrazuje taki zbiór działań (kombinację sygnałów sterujących), że działania powodowane przez różne pola mogą następować jednocześnie.
- Zdefiniowanie każdego pola w taki sposób, żeby alternatywne działania określane przez to pole wzajemnie się wykluczały. Oznacza to, że tylko jedno spośród działań precyzowanych przez dane pole może następować w określonej chwili.

Możliwe są dwa podejścia do podziału kodowanego mikrorozkazu na pola: funkcjonalne oraz według zasobów. *Funkcjonalna metoda kodowania* identyfikuje funkcje wewnątrz maszyny i przypisuje pola rodzajom funkcji. Jeśli na przykład mogą występować różne źródła danych przenoszonych do akumulatora, jedno z pól może być



(a) Kodowanie bezpośrednie



(b) Kodowanie pośrednie

Rysunek 17.11. Kodowanie mikrorozkazu

Proste transfery rejestrowe

Kod	Operacja
000 | 000 | _ _	MDR ← Rejestr
000 | 001 | _ _	Rejestr ← MDR
000 | 010 | _ _	MAR ← Rejestr

Wybór rejestru

Operacje pamięci

Kod	Operacja
001 | 000 | 00	Odczyt
001 | 001 | 00	Zapis

Specjalne operacje szeregowania

Kod	Operacja
010 | 000 | 00	CSAR← Zdekodowany MDR
010 | 001 | 00	SCAR ← Stała (w następnym bajcie)
010 | 010 | 00	Pominięcie

Operacje ALU

Kod	Operacja
011 | 000 | _ _	ACC ← ACC + Rejestr
011 | 001 | _ _	ACC ← ACC – Rejestr
011 | 010 | _ _	ACC ← Rejestr
011 | 011 | _ _	Rejestr ← ACC
011 | 100 | _ _	ACC ← Rejestr + 1

Wybór rejestru

(a) Repertuar rozkazów pionowych

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Pole 1 2 3 4 5 6

Definicja pola:
1 – transfer rejestrowy
2 – operacja pamięci
3 – operacja szeregowania
4 – operacja ALU
5 – wybór rejestru
6 – stała

(b) Format rozkazu poziomego

Rysunek 17.12. Alternatywne formaty mikrorozkazu prostego komputera

przypisane do tego celu, przy czym każdy kod określa inne źródło. *Kodowanie według zasobów* opiera się na postrzeganiu maszyny jako zbioru niezależnych zasobów i polega na przypisaniu każdemu z nich (np. pamięci, wejściu-wyjściu, ALU) jednego pola.

Innym aspektem kodowania jest to, czy jest ono bezpośrednie, czy pośrednie (rys. 17.11b). W przypadku kodowania pośredniego jedno pole jest używane do określania interpretacji drugiego pola. Rozważmy na przykład ALU, która może przeprowadzić osiem różnych operacji arytmetycznych i osiem różnych operacji przesuwania. Pole 1-bitowe mogłoby być użyte do wskazywania, czy ma być przeprowadzona operacja arytmetyczna, czy przesuwania; pole 3-bitowe wskazywałoby operację. Metoda ta na ogół implikuje dwa poziomy dekodowania, co zwiększa opóźnienia propagacji.

Na rysunku 17.12 jest podany prosty przykład tych koncepcji. Załóżmy istnienie procesora z jednym akumulatorem i kilkoma rejestrami wewnętrznymi, takimi jak licznik programu i rejestr tymczasowy wyjścia ALU. Na rysunku 17.12a jest pokazane rozwiązanie pionowe. Pierwsze 3 bity wskazują rodzaj operacji, następne 3 są kodem operacji, a ostatnie 2 służą do wybierania rejestru wewnętrznego. Na rysunku 17.12b są przedstawione rozwiązania bardziej poziome, chociaż w dalszym ciągu jest stosowane kodowanie. W tym przypadku różne pola dotyczą różnych funkcji.

Wykonywanie mikrorozkazów w LSI-11

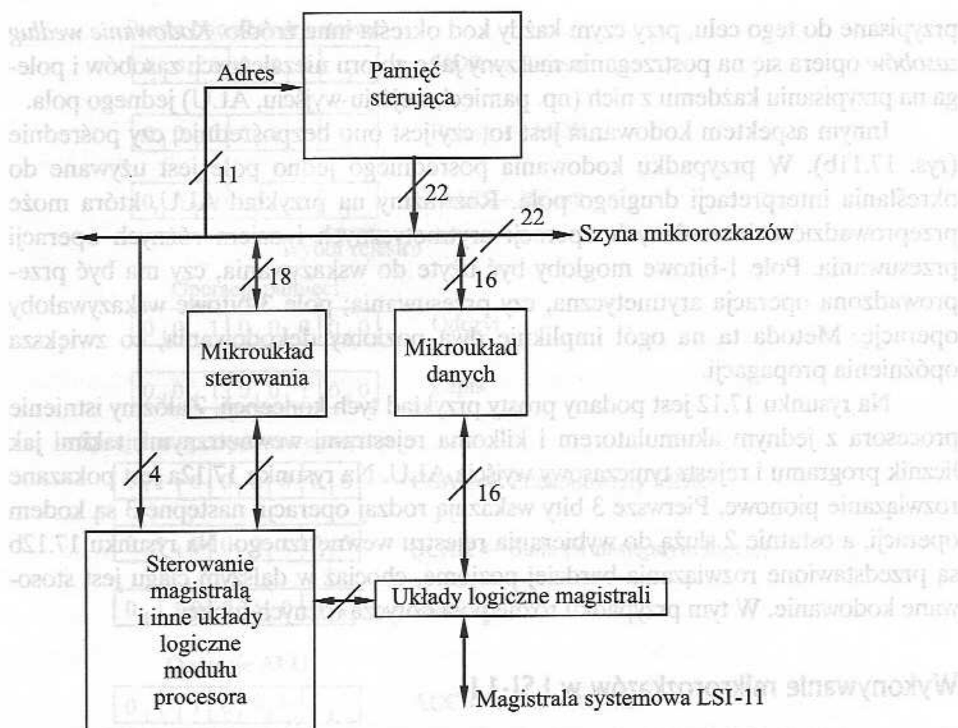
Procesor LSI-11 [SEBE76] stanowi dobry przykład pionowego formatu mikrorozkazu. Najpierw zapoznamy się z organizacją jednostki sterującej, a następnie z formatem mikrorozkazu.

Organizacja jednostki sterującej LSI-11

Procesor LSI-11 jest pierwszym modelem rodziny PDP-11 oferowanym jako procesor na pojedynczej płytce drukowanej. Na płytce znajdują się trzy mikroukłady LSI, magistrala wewnętrzna, określana jako *magistrala mikrorozkazu*, oraz pewne dodatkowe układy logiczne interfejsu.

Na rysunku 17.13 widać organizację procesora LSI-11 w postaci uproszczonej. Występują tu trzy mikroukłady: mikroukład danych, mikroukład sterowania i pamięć sterowania. Mikroukład danych zawiera 8-bitową ALU, 26 rejestrów 8-bitowych i pamięć kilku kodów warunkowych. Szesnaście rejestrów posłużyło do wdrożenia 8 16-bitowych rejestrów ogólnego przeznaczenia PDP-11. Pozostałe obejmują rejestr słowa stanu programu, rejestr adresu pamięci (MAR) i buforowy rejestr pamięci. Ponieważ ALU przeprowadza w określonej chwili operacje jedynie na 8 bitach, do wdrożenia 16-bitowej operacji arytmetycznej procesora PDP-11 są wymagane dwa przejścia przez ALU. Jest to sterowane mikroprogramem.

Mikroukład (lub mikroukłady) pamięci sterowania zawiera pamięć sterowania o szerokości 22 bitów. Mikroukład sterowania zawiera też układy logiczne do szeregowania i wykonywania mikrorozkazów oraz rejestr adresu sterowania, rejestr danych sterowania i kopię rejestru rozkazu maszynowego.



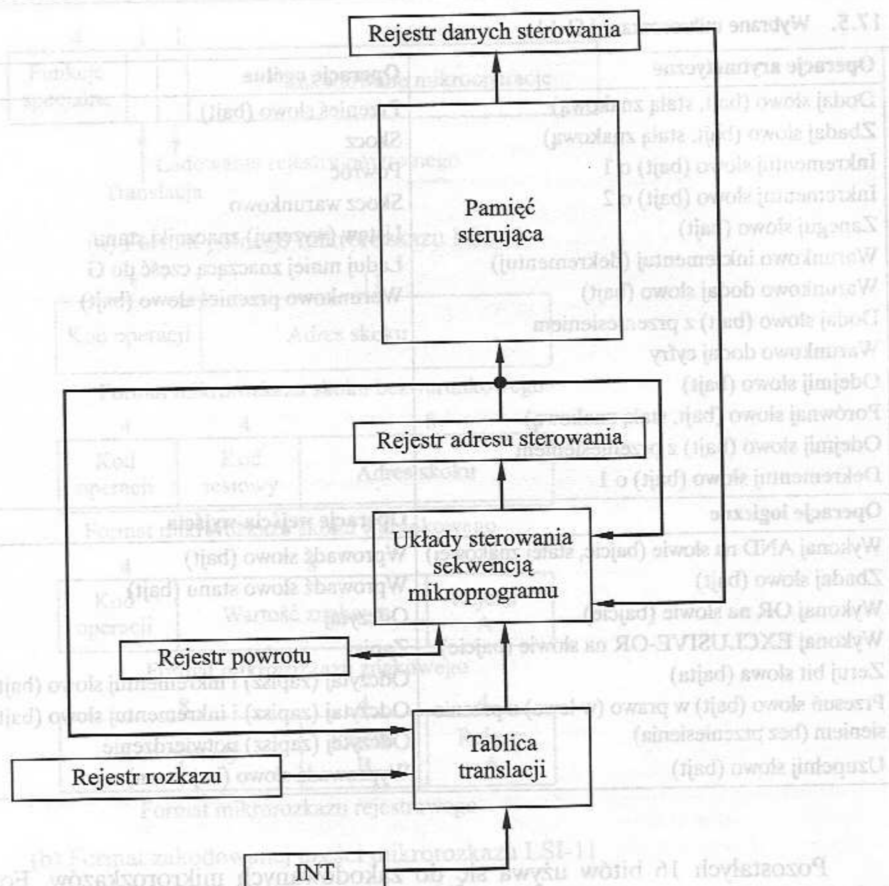
Rysunek 17.13. Uproszczony schemat blokowy procesora LSI-11

Magistrala MIB łączy wszystkie podzespoły. Podczas pobierania mikrorozkazu mikroukład sterowania generuje 11-bitowy adres i umieszcza go na magistrali MIB. Z pamięci sterowania jest pobierany 22-bitowy mikrorozkaz, który jest lokowany na magistrali MIB. Do mikroukładu danych jest kierowanych 16 najmniej znaczących bitów, podczas gdy 18 najmniej znaczących bitów trafia do mikroukładu sterowania. Specjalne funkcje na płytce procesora są sterowane za pomocą 4 najbardziej znaczących bitów.

Na rysunku 17.14 jest przedstawiony nadal uproszczony, jednak bardziej szczegółowy obraz jednostki sterującej LSI-11 (na rysunku pominięto granice poszczególnych mikroukładów). Schemat szeregowania adresów, opisany w podrozdz. 17.2, został wdrożony w postaci dwóch modułów. Ogólne sterowanie sekwencją zapewnia moduł sterowania sekwencją mikrorozkazów, który ma możliwość zwiększania stanu rejestru adresu mikrorozkazu i przeprowadzania rozgałęzień bezwarunkowych. Inne formy obliczania adresu są przeprowadzane przez oddzielną tablicę translacji. Tablica ta to układ kombinacyjny, który generuje adres oparty na: mikrorozkazie, rozkazie maszynowym, stanie licznika mikroprogramu i stanie rejestru przerwań.

Tablica translacji jest używana w następujących okolicznościach:

- gdy kod operacji jest używany do rozpoczęcia podprogramu standardowego;
- gdy bity trybu adresowania w mikrorozkazie są testowane w celu przeprowadzenia właściwego adresowania;



Rysunek 17.14. Organizacja jednostki sterującej LSI-11

- gdy okresowo są testowane warunki przerwania;
- gdy analizowane są mikrorozkazy rozgałęzienia warunkowego.

Format mikrorozkazu LSI-11

Format mikrorozkazu LSI-11 można określić jako krańcowo pionowy. Rozkaz ma szerokość zaledwie 22 bitów. Lista mikrorozkazów ściśle zastępuje listę rozkazów maszynowych PDP-11. Intencją projektu była optymalizacja wydajności jednostki sterującej w ramach ograniczeń stwarzanych przez rozwiązanie pionowe, łatwe do programowania. W tabeli 17.5 są wymienione niektóre mikrorozkazy LSI-11.

Na rysunku 17.15 jest pokazany 22-bitowy format mikrorozkazu LSI-11. Specjalnymi funkcjami sterują 4 najbardziej znaczące bity. Bit translacji zezwala tablicy translacji na sprawdzenie przerwań zawieszonych. Bit ładowania rejestru powrotu jest używany na zakończenie podprogramu standardowego do powodowania ładowania adresu następnego mikrorozkazu z rejestru powrotu.

Tabela 17.5. Wybrane mikrorozkazy LSI-11

Operacje arytmetyczne	Operacje ogólne
Dodaj słowo (bajt, stałą znakową)	Przenieś słowo (bajt)
Zbadaj słowo (bajt, stałą znakową)	Skocz
Inkrementuj słowo (bajt) o 1	Powróć
Inkrementuj słowo (bajt) o 2	Skocz warunkowo
Zaneguj słowo (bajt)	Ustaw (wyzeruj) znaczniki stanu
Warunkowo inkrementuj (dekrementuj)	Ładuj mniej znaczącą część do G
Warunkowo dodaj słowo (bajt)	Warunkowo przenieś słowo (bajt)
Dodaj słowo (bajt) z przeniesieniem	
Warunkowo dodaj cyfry	
Odejmij słowo (bajt)	
Porównaj słowo (bajt, stałą znakową)	
Odejmij słowo (bajt) z przeniesieniem	
Dekrementuj słowo (bajt) o 1	
Operacje logiczne	**Operacje wejścia-wyjścia**
Wykonaj AND na słowie (bajcie, stałej znakowej)	Wprowadź słowo (bajt)
Zbadaj słowo (bajt)	Wprowadź słowo stanu (bajt)
Wykonaj OR na słowie (bajcie)	Odczytaj
Wykonaj EXCLUSIVE-OR na słowie (bajcie)	Zapisz
Zeruj bit słowa (bajta)	Odczytaj (zapisz) i inkrementuj słowo (bajt) o 1
Przesuń słowo (bajt) w prawo (w lewo) z przeniesieniem (bez przeniesienia)	Odczytaj (zapisz) i inkrementuj słowo (bajt) o 2
	Odczytaj (zapisz) potwierdzenie
Uzupełnij słowo (bajt)	Wyprowadź słowo (bajt, stan)

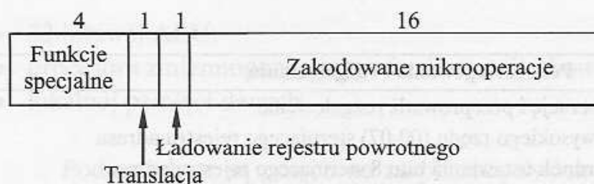
Pozostałych 16 bitów używa się do zakodowanych mikrorozkazów. Format jest tu podobny do rozkazu maszynowego, z kodem operacji o zmiennej długości i z jednym lub z wieloma argumentami.

Wykonywanie mikrorozkazów w IBM 3033

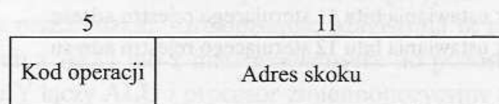
Standardowa pamięć sterowania IBM 3033 składa się z 4K słów. Pierwsza połowa z nich (0000÷07FF) zawiera mikrorozkazy 108-bitowe, podczas gdy pozostałe (0800÷0FFF) są używane do przechowywania mikrorozkazów 126-bitowych. Format jest przedstawiony na rys. 17.16. Chociaż jest on raczej poziomy, mimo to szeroko jest stosowane kodowanie. Główne pola tego formatu są podane w tabeli 17.6.

Dane na wejścia ALU są podawane z czterech wyspecjalizowanych, niewidzialnych dla użytkownika rejestrów (A, B, C i D). Format mikrorozkazu zawiera pola służące do ładowania tych rejestrów z rejestrów widzialnych dla użytkownika, służących do wykonywania działań ALU oraz określania rejestru widzialnego dla użytkownika służącego do przechowania wyniku. Istnieją również pola służące do ładowania i zapisu danych między rejestrami a pamięcią.

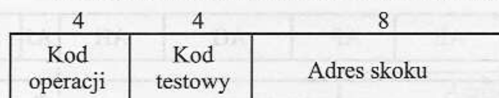
Mechanizm szeregowania w IBM 3033 omówiliśmy w podrozdz. 17.2.



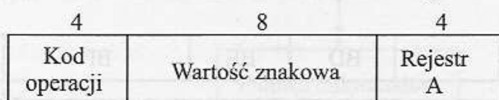
(a) Format pełnego mikrorozkazu LSI-11



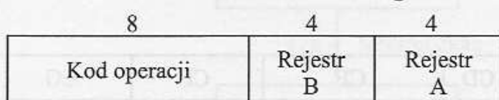
Format mikrorozkazu skoku bezwarunkowego



Format mikrorozkazu skoku warunkowego



Format mikrorozkazu znakowego



Format mikrorozkazu rejestrowego

(b) Format zakodowanej części mikrorozkazu LSI-11

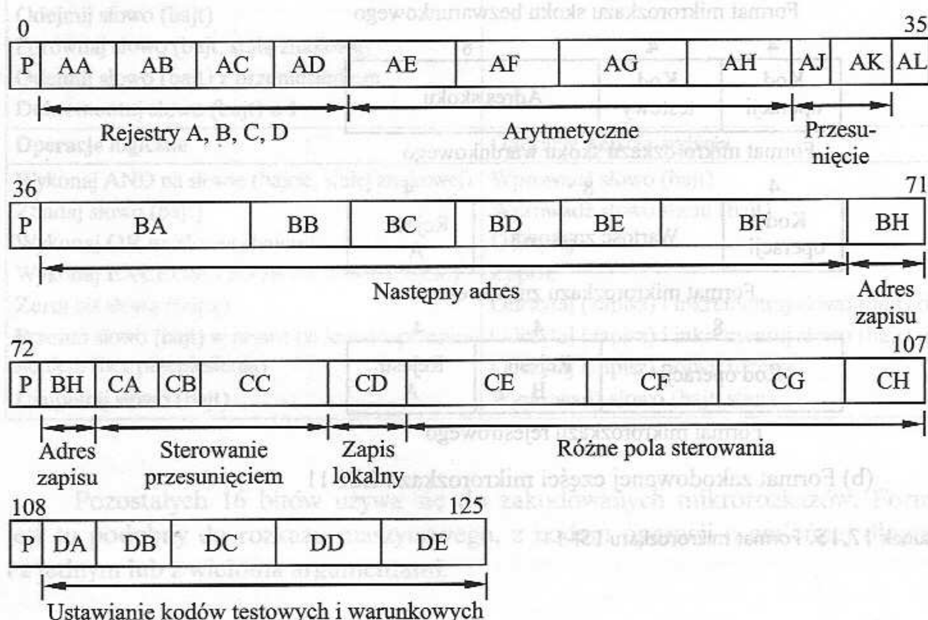
Rysunek 17.15. Format mikrorozkazu LSI-11

Tabela 17.6. Pola sterujące mikrorozkazów IBM 3033

	Pola sterujące ALU
AA(3)	Ładuj rejestr A z jednego spośród rejestrów danych
AB(3)	Ładuj rejestr B z jednego spośród rejestrów danych
AC(3)	Ładuj rejestr C z jednego spośród rejestrów danych
AD(3)	Ładuj rejestr D z jednego spośród rejestrów danych
AE(4)	Skieruj określone bity A do ALU
AF(4)	Skieruj określone bity B do ALU
AG(5)	Określa operację arytmetyczną ALU na wejściu A
AH(4)	Określa operację arytmetyczną ALU na wejściu B
AJ(1)	Określa wejście D lub B do ALU po stronie B
AK(4)	Skieruj wyjście arytmetyczne do przesuwnika
CB(1)	Aktywuj przesuwnik
CC(5)	Określa funkcje logiczne i przeniesienia
CE(7)	Określa wielkość przesunięcia
CA(3)	Ładuj rejestr F

Tabela 17.6 (cd.)

Pola szeregowania i rozgałęziania	
AL(1)	Zakończ operację i przeprowadź rozgałęzienie
BA(8)	Ustaw bity wysokiego rzędu (00-07) sterującego rejestru adresu
BB(4)	Określa warunek ustawiania bitu 8 sterującego rejestru adresu
BC(4)	Określa warunek ustawiania bitu 9 sterującego rejestru adresu
BD(4)	Określa warunek ustawiania bitu 10 sterującego rejestru adresu
BE(4)	Określa warunek ustawiania bitu 11 sterującego rejestru adresu
BF(4)	Określa warunek ustawiania bitu 12 sterującego rejestru adresu



Rysunek 17.16. Format mikrorozkazu IBM 3033

17.4. Moduł TI 8800

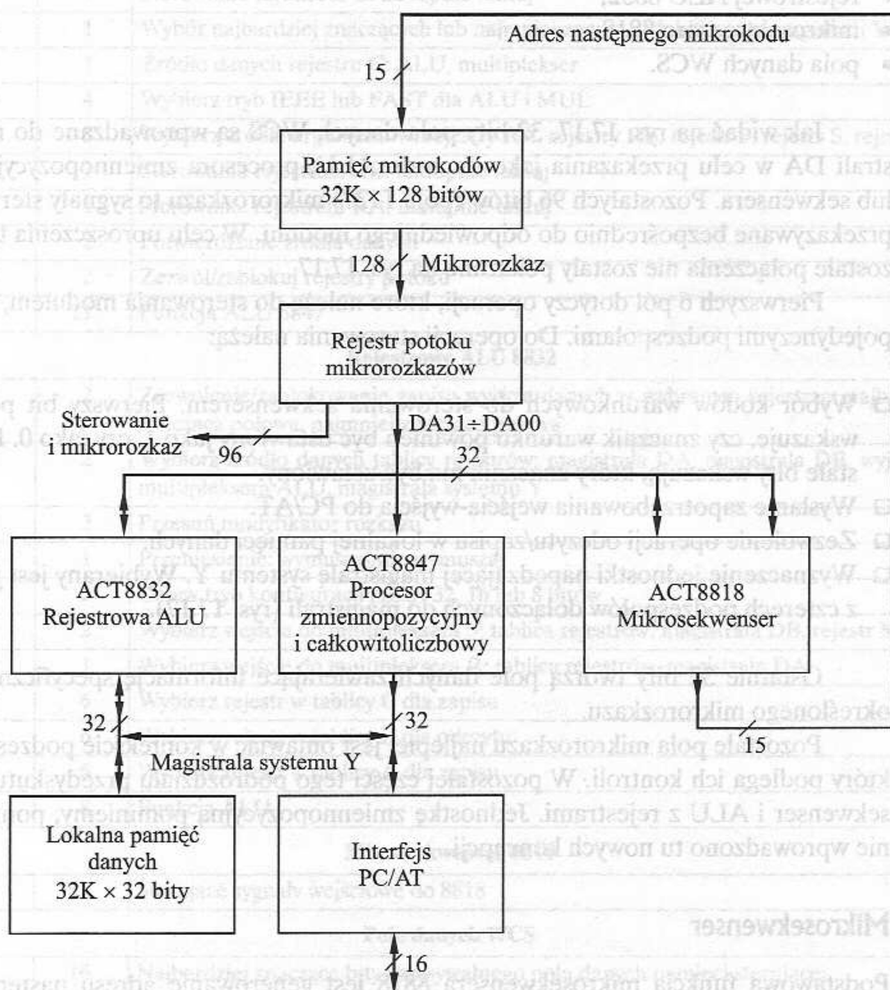
Moduł SDB 8800 (skrót od *Software Development Board,* moduł do opracowywania oprogramowania) firmy Texas Instruments jest mikroprogramowanym 32-bitowym modułem komputerowym. W jego skład wchodzi zapisywalna pamięć sterowania, wdrożona w postaci RAM zamiast ROM. Nie osiąga on szybkości ani gęstości systemu mikroprogramowanego z pamięcią sterowania typu ROM. Jest jednak użyteczny do opracowywania prototypów i do celów edukacyjnych.

Moduł SDB 8800 składa się z następujących podzespołów (rys. 17.17):

- pamięci mikrorozkazów;
- mikrosekwensera;

- 32-bitowej ALU;
- procesora zmiennopozycyjnego i całkowitoliczbowego;
- lokalnej pamięci danych.

Podzespoły te są połączone za pomocą dwóch magistrali. Magistrala DA dostarcza danych z pola danych mikrorozkazu do ALU, procesora zmiennopozycyjnego lub mikrosekwensera. W tym ostatnim przypadku dane obejmują adres, który ma być użyty przez rozkaz rozgałęziania. Magistrala ta może również służyć do przekazania danych z ALU lub z mikrosekwensera do pozostałych podzespołów. Magistrala systemu Y łączy ALU i procesor zmiennopozycyjny z pamięcią lokalną i z modułami zewnętrznymi poprzez interfejs PC.



Rysunek 17.17. Schemat blokowy modułu TI 8800

Moduł jest dopasowany do komputerów macierzystych kompatybilnych z IBM PC. Komputer macierzysty stanowi odpowiednią platformę do asemblacji mikroprogramu i jego uruchomienia.

Format mikrorozkazu

Format mikrorozkazu modułu 8800 składa się ze 128 bitów podzielonych na 30 pól funkcjonalnych, co widać w tabeli 17.7. Każde pole składa się z jednego lub wielu bitów. Pola są zgrupowane w pięć głównych kategorii:

- sterowania modułu;
- mikroukładu procesora zmiennopozycyjnego i całkowitoliczbowego 8847;
- rejestrowej ALU 8832;
- mikrosekwensera 8818;
- pola danych WCS.

Jak widać na rys. 17.17, 32 bity pola danych WCS są wprowadzane do magistrali DA w celu przekazania jako dane do ALU, procesora zmiennopozycyjnego lub sekwensera. Pozostałych 96 bitów (pola 1÷27) mikrorozkazu to sygnały sterujące przekazywane bezpośrednio do odpowiedniego modułu. W celu uproszczenia te pozostałe połączenia nie zostały pokazane na rys. 17.17.

Pierwszych 6 pól dotyczy operacji, które należą do sterowania modułem, a nie pojedynczymi podzespołami. Do operacji sterowania należą:

- Wybór kodów warunkowych do sterowania sekwenserem. Pierwszy bit pola 1 wskazuje, czy znacznik warunku powinien być ustawiony jako 1, czy jako 0. Pozostałe bity wskazują, który znacznik ma być ustawiony.
- Wysłanie zapotrzebowania wejścia-wyjścia do PC/AT.
- Zezwolenie operacji odczytu/zapisu w lokalnej pamięci danych.
- Wyznaczenie jednostki napędzającej magistralę systemu Y. Wybierany jest jeden z czterech podzespołów dołączonych do magistrali (rys. 17.17).

Ostatnie 32 bity tworzą pole danych zawierające informację specyficzną dla określonego mikrorozkazu.

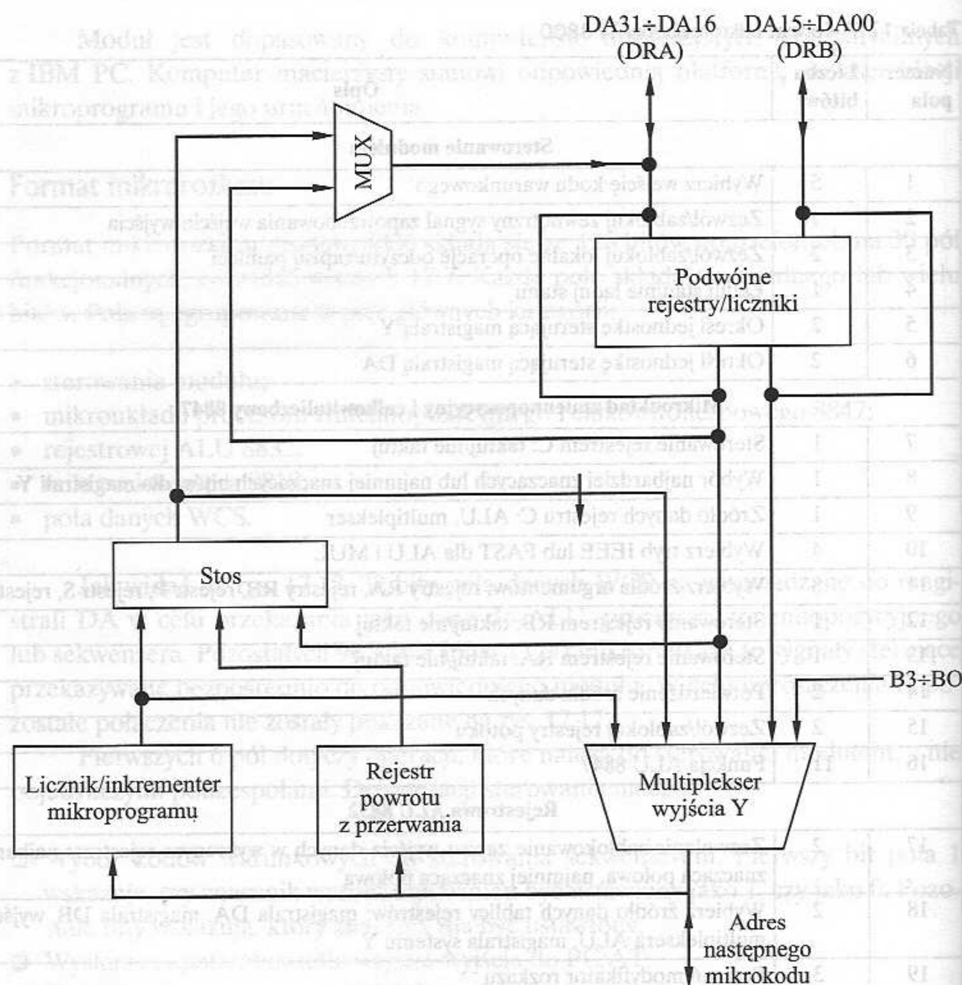
Pozostałe pola mikrorozkazu najlepiej jest omawiać w kontekście podzespołu, który podlega ich kontroli. W pozostałej części tego podrozdziału przedyskutujemy sekwenser i ALU z rejestrami. Jednostkę zmiennopozycyjną pominiemy, ponieważ nie wprowadzono tu nowych koncepcji.

Mikrosekwenser

Podstawową funkcją mikrosekwensera 8818 jest generowanie adresu następnego mikrorozkazu w mikroprogramie. Do pamięci mikrorozkazów jest doprowadzany 15-bitowy adres (rys. 17.17).

Tabela 17.7. Format mikrorozkazów TI 8800

Numer pola	Liczba bitów	Opis
		Sterowanie modułem
1	5	Wybierz wejście kodu warunkowego
2	1	Zezwól/zablokuj zewnętrzny sygnał zapotrzebowania wejścia-wyjścia
3	2	Zezwól/zablokuj lokalne operacje odczytu/zapisu pamięci
4	1	Ładuj stan/nie ładuj stanu
5	2	Określ jednostkę sterującą magistralą Y
6	2	Określ jednostkę sterującą magistralą DA
		Mikroukład zmiennopozycyjny i całkowitoliczbowy 8847
7	1	Sterowanie rejestrem C: taktuj/nie taktuj
8	1	Wybór najbardziej znaczących lub najmniej znaczących bitów dla magistrali Y
9	1	Źródło danych rejestru C: ALU, multiplekser
10	4	Wybierz tryb IEEE lub FAST dla ALU i MUL
11	8	Wybierz źródła argumentów: rejestry RA, rejestry RB, rejestr P, rejestr S, rejestr C
12	1	Sterowanie rejestrem RB: taktuj/nie taktuj
13	1	Sterowanie rejestrem RA: taktuj/nie taktuj
14	2	Potwierdzenie źródła danych
15	2	Zezwól/zablokuj rejestry potoku
16	11	Funkcja ALU 8847
		Rejestrowa ALU 8832
17	2	Zezwolenie/zablokowanie zapisu wyjścia danych w wybranym rejestrze: najbardziej znacząca połowa, najmniej znacząca połowa
18	2	Wybierz źródło danych tablicy rejestrów: magistrala DA, magistrala DB, wyjście Y multipleksera ALU, magistrala systemu Y
19	3	Przesuń modyfikator rozkazu
20	1	Przeniesienie: wymuszaj/nie wymuszaj
21	2	Ustaw tryb konfiguracji ALU: 32, 16 lub 8 bitów
22	2	Wybierz wejście do multipleksera S: tablica rejestrów, magistrala DB, rejestr MQ
23	1	Wybierz wejście do multipleksera R: tablica rejestrów, magistrala DA
24	6	Wybierz rejestr w tablicy C dla zapisu
25	6	Wybierz rejestr w tablicy B dla odczytu
26	6	Wybierz rejestr w tablicy A dla zapisu
27	8	Funkcja ALU
		Mikrosekwenser 8818
28	12	Sterujące sygnały wejściowe do 8818
		Pole danych WCS
29	16	Najbardziej znaczące bity zapisywalnego pola danych pamięci sterującej
30	16	Najmniej znaczące bity zapisywalnego pola danych pamięci sterującej



Rysunek 17.18. Mikrosekwenser TI 8818

Następny adres może być wybrany z jednego z pięciu źródeł, którymi są:

1. Rejestr licznika mikroprogramu (MPC) używany do powtarzania (ponownego użycia tego samego adresu) i kontynuowania (zwiększania adresu o 1) rozkazów.
2. Stos, który obsługuje wywołania podprogramów standardowych, a także pętle iteracyjne i powroty z przerwań.
3. Porty DRA i DRB, stanowiące dwie dodatkowe ścieżki z urządzeń zewnętrznych, przez które mogą być generowane adresy mikroprogramu. Porty te są połączone odpowiednio z najbardziej znaczącymi i z najmniej znaczącymi bitami magistrali DA. Dzięki temu sekwenser może otrzymać adres następnego rozkazu z pola danych WCS bieżącego mikrorozkazu lub na podstawie wyniku obliczonego przez ALU.

4. Liczniki rejestrowe RCA i RCB, które mogą być użyte do przechowywania adresu.
5. Zewnętrzne wejście do dwukierunkowego portu Y, wspierające przerwania zewnętrzne.

Na rysunku 17.18 widać blokowy schemat logiczny mikrosekwensera 8818. Podzespół ten składa się z następujących głównych części funkcjonalnych:

- 16-bitowego licznika mikroprogramu (MPC) zbudowanego z rejestru i układu zwiększania stanu rejestru.
- Dwóch liczników rejestrowych, RCA i RCB, służących do liczenia pętli i iteracji, przechowywania adresów rozgałęzień lub sterowania przyrządami zewnętrznymi.
- Stosu o pojemności 65 słów 16-bitowych, umożliwiającego wywoływanie podprogramów standardowych i przerwania.
- Rejestru powrotu z przerwań oraz układu zezwalania wyjścia Y, służącego do przetwarzania przerwań na poziomie mikrorozkazu.
- Multipleksera wyjścia Y, za pośrednictwem którego następny adres może być wybrany z MPC, RCA, RCB, z magistrali zewnętrznych DRA i DRB lub ze stosu.

Rejestry/liczniki

Rejestry RCA i RCB mogą być ładowane przez magistralę DA albo z bieżącego mikrorozkazu, albo z wyjścia ALU. Ich zawartości mogą być używane jako liczniki do sterowania przebiegiem wykonywania i mogą być automatycznie dekrementowane przy dostępie. Zawartości te mogą być również używane jako adresy mikrorozkazów, które mają być doprowadzone do multipleksera wyjścia Y. Wspierane jest niezależne sterowanie oboma rejestrami podczas cyklu mikrorozkazu, z wyjątkiem jednoczesnego zmniejszania zawartości obu rejestrów.

Stos

Stos umożliwia wiele poziomów zagnieżdżonych wywołań lub przerwań i może być używany do obsługi rozgałęzień i pętli. Pamiętajmy, że operacje te odnoszą się tylko do jednostki sterującej, a nie do całego procesora, i że używane adresy są adresami mikrorozkazów w pamięci sterowania.

Możliwych jest sześć operacji stosu:

1. Kasowanie, które ustawia wskaźnik stosu na zero, opróżniając stos.
2. Zdejmowanie ze stosu, które powoduje dekrementowanie wskaźnika stosu.
3. Umieszczanie na stosie zawartości MPC, rejestru powrotów z przerwań lub magistrali DRA i jednoczesne inkrementowanie wskaźnika stosu.
4. Odczyt adresu wskazanego przez wskaźnik odczytu i udostępnienie go multiplekserowi wyjścia Y.
5. Wstrzymanie, które powoduje, że adres wskaźnika stosu pozostaje niezmieniony.
6. Ładowanie wskaźnika stosu za pomocą 7 najmniej znaczących bitów DRA.

Sterowanie mikrosekwenserem

Mikrosekwenser jest sterowany przede wszystkim przez 12-bitowe pole bieżącego mikrorozkazu – pole 28 (tabela 17.7). Pole to składa się z następujących pól cząstkowych:

- **OSEL (1 bit)** – pole wyboru wyjścia. Określa, która wartość zostanie umieszczona na wyjściu multipleksera, połączonego z magistralą DRA (górny lewy róg rys. 17.18). Wyjście jest wybierane w ten sposób, że pochodzi albo ze stosu, albo z rejestru RCA. Z magistrali DRA jest on następnie podawany albo na wejście multipleksera wyjścia Y, albo do rejestru RCA.
- **SELDR (1 bit)** – pole wyboru magistrali DR. Jeśli jest ustawione na 1, to oznacza wybór zewnętrznej magistrali DA jako wejścia do magistrali DRA/DRB. Jeśli jest ustawione na 0, to oznacza wybór wyjścia multipleksera DRA do magistrali DRA (sterowane przez OSEL) oraz skierowanie zawartości rejestru RCB do magistrali DRB.
- **ZEROIN (1 bit).** Używany do wskazywania rozgałęzienia warunkowego. Zachowanie mikrosekwensera będzie następnie zależało od kodu warunkowego wybranego w polu 1 (tabela 17.7).
- **RC2÷RC0 (3 bity)** – bity sterujące rejestrami. Określają zmianę zawartości rejestrów RCA i RCB. Zawartość rejestrów albo może pozostać bez zmian, albo może być dekrementowana, albo wreszcie rejestry mogą być ładowane z magistrali DRA/DRB.
- **S2÷S0 (3 bity)** – bity sterujące stosem. Określają, jaka operacja stosu ma być wykonywana.
- **MUX2÷MUX0** – bity sterowania wyjściem. W połączeniu z kodem warunkowym (jeśli jest używany) sterują multiplekserem wyjścia Y, określając w ten sposób adres następnego mikrorozkazu. Multiplekser może wybrać swoje wyjście ze stosu, rejestrów DRA, DRB lub MPC.

Bity te mogą być ustawiane indywidualnie przez programistę. Zwykle jednak tak się nie robi. Zamiast tego programista używa mnemoników odpowiadających potrzebnej kombinacji bitów. W tabeli 17.8 jest wymienionych 15 mnemoników dla pola 28. Asembler mikroprogramu zamienia je na odpowiednie wzory bitowe.

Jako przykład niech posłuży rozkaz INC88181, który jest używany do wybrania następnego mikrorozkazu jako kolejnego w sekwencji, jeśli bieżąco wybranym kodem warunkowym jest 1. Z tabeli 17.8 mamy

```
INC88181 = 000000111110
```

co dekoduje się bezpośrednio na

- **OSEL = 0.** Wybranie RCA na wyjściu DRA multipleksera MUX; w tym przypadku wybór ten jest nieistotny.
- **SELDR = 0.** Jak wyżej; jest to nieistotne w przypadku tego rozkazu.

- **ZEROIN = 0.** W połączeniu z wartością MUX wskazuje, że nie nastąpi rozgałęzienie.
- **R = 000.** Zachowanie bieżącej wartości RA i RC.
- **S = 111.** Zachowanie bieżącego stanu stosu.
- **MUX = 110.** Wybranie MPC, jeśli kod warunkowy = 1; wybranie DRA, jeśli kod warunkowy = 0.

Tabela 17.8. Bity mikrorozkazu mikrosekwensera TI 8818 (pole 28)

Mnemonik	Wartość	Opis
RST8818	000000000110	Rozkaz sprowadzania do stanu wyjściowego
BRA88181	011000111000	Rozgałęziaj do rozkazu DRA
BRA88180	010000111110	Rozgałęziaj do rozkazu DRA
INC88181	000000111110	Kontynuuj rozkaz
INC88180	001000001000	Kontynuuj rozkaz
CAL88181	010000110000	Skocz do podprogramu pod adresem określonym przez DRA
CAL88180	010000101110	Skocz do podprogramu pod adresem określonym przez DRA
RET8818	000000011010	Powróć z podprogramu
PUSH8818	000000110111	Umieść adres powrotu z przerwania na stosie
POP8818	100000010000	Powróć z przerwania
LOADDRA	000010111110	Ładuj licznik DRA z magistrali DA
LOADDRB	000110111110	Ładuj licznik DRB z magistrali DA
LOADDRAB	000110111100	Ładuj DRA/DRB
DECRDRA	010001111100	Dekrementuj licznik DRA i rozgałęziaj, jeśli nie zero
DECRDRB	010101111100	Dekrementuj licznik DRB i rozgałęziaj, jeśli nie zero

Rejestrowa ALU

Moduł 8832 jest 32-bitową ALU z 64 rejestrami, która może być konfigurowana w postaci czterech 8-bitowych ALU, dwóch 16-bitowych ALU lub jednej 32-bitowej ALU.

Moduł 8832 jest sterowany przez 39 bitów w polach mikrorozkazu (pola 17÷27 w tabeli 17.7); są one doprowadzane do ALU jako sygnały sterujące. Ponadto, jak widać na rys. 17.17, moduł 8832 jest połączony z 32-bitową magistralą DA i z 32-bitową magistralą systemu Y. Wejścia z magistrali DA mogą być doprowadzone jednocześnie jako wejścia danych do tablicy rejestrów obejmującej 64 słowa oraz do modułu logicznego ALU. Wejście z magistrali systemu Y jest doprowadzone do modułu logicznego ALU. Wyniki operacji ALU oraz przesunięcia są kierowane do magistrali DA lub magistrali systemu Y. Mogą one być również dostarczone do wewnętrznej tablicy rejestrów.

Trzy 6-bitowe porty adresowe umożliwiają jednoczesne pobieranie dwóch argumentów i zapisywanie argumentu w tablicy rejestrów. Przesuwnik MQ i rejestr MQ mogą być również skonfigurowane jako działające niezależnie w celu wdrożenia operacji 8- i 16-bitowych o podwójnej dokładności oraz 32-bitowej operacji przesunięcia.

Pola od 17 do 26 każdego mikrorozkazu sterują sposobem przepływu danych wewnątrz modułu 8832 oraz między 8832 a otoczeniem. Pola te są następujące:

17. **Zezwolenie zapisu.** Dwa bity tego pola określają zapis 32 bitów, 16 najbardziej znaczących bitów lub 16 najmniej znaczących bitów, lub też brak zapisu w tablicy rejestrów. Rejestr docelowy jest określany w polu 24.
18. **Wybór źródła danych tablicy rejestrów.** Jeśli następuje zapis w tablicy rejestrów, to dwa bity tego pola określają źródło: magistralę DA, magistralę DB, wyjście ALU lub magistralę systemu Y.
19. **Modyfikator rozkazu przesunięcia.** Określa opcje dotyczące dostarczania bitów wypełniających oraz odczytu bitów przesuwanych podczas operacji przesunięcia.
20. **Przeniesienie do ALU.** Bit ten określa, czy następuje przeniesienie bitu do ALU w danej operacji.
21. **Rodzaj konfiguracji ALU.** Moduł 8832 może być skonfigurowany jako jedna 32-bitowa ALU, dwie 16-bitowe ALU lub cztery 8-bitowe ALU.
22. **Wejście S.** Wejścia modułu logicznego ALU są obsługiwane przez dwa wewnętrzne multipleksery oznaczane jako S i R. Pole to służy do wybrania wejścia obsługiwanego przez multiplekser S: z tablicy rejestrów, magistrali DB lub rejestru MQ. Rejestr źródłowy jest wybierany w polu 25.
23. **Wejście R.** Służy do wybrania wejścia obsługiwanego przez multiplekser R: z tablicy rejestrów lub z magistrali DA.
24. **Rejestr docelowy.** Określa adres rejestru w tablicy rejestrów, który ma być użyty dla argumentu docelowego.
25. **Rejestr źródłowy.** Określa adres rejestru w tablicy rejestrów, który ma być użyty dla argumentu źródłowego, dostarczany przez multiplekser S.
26. **Rejestr źródłowy.** Określa adres rejestru w tablicy rejestrów, który ma być użyty dla argumentu źródłowego, dostarczany przez multiplekser R.

Wreszcie pole 27 zawiera 8-bitowy kod operacji określający działanie arytmetyczne lub logiczne, które ma być realizowane przez ALU. W tabeli 17.9 są wymienione operacje, które mogą być realizowane.

Jako przykład kodowania pól 17÷27, rozważmy rozkaz dodania zawartości rejestru 1 do rejestru 2 i umieszczenia wyniku w rejestrze 3. Symbolicznym rozkazem jest

```
CONT11 [17], WELH, SELRYFYMX, [24], R3, R2, R1, PASS+ADD
```

Asembler przekształci to na odpowiedni wzór bitowy. Poszczególne składniki rozkazu mogą być opisane następująco:

- CONT11 jest podstawowym rozkazem NOP.
- Pole [17] jest zmieniane na WELH (zezwolenie zapisu, najmniej znaczące i najbardziej znaczące), dzięki czemu następuje wpis do rejestru 32-bitowego.
- Pole [18] jest zmieniane na SELRFYMX w celu wybrania wyjścia ALU Y MUX.
- Pole [24] jest zmieniane w celu wyznaczenia rejestru R3 jako docelowego.

- Pole [25] jest zmieniane w celu wyznaczenia rejestru R2 jako jednego z rejestrów źródłowych.
- Pole [26] jest zmieniane w celu wyznaczenia rejestru R1 jako jednego z rejestrów źródłowych.
- Pole [27] jest zmieniane w celu określenia operacji ADD (dodawania) ALU. Rozkazem dla przesuwnika ALU jest PASS; wobec tego wartość z wyjścia ALU nie jest przesuwana przez przesuwnik.

Można tu podać kilka uwag na temat notacji symbolicznej. Nie jest konieczne określanie numeru pola w przypadku kolejnych pól. Znaczy to, że

```
CONT11 [17], WELH, [18], SELRFYMX
```

może być zapisane jako

```
CONT11 [17], WELH, SELRFYMX
```

ponieważ SELRFYMX jest w polu [18].

Rozkazy ALU z grupy 1 w tabeli 17.9 muszą zawsze być używane w połączeniu z grupą 2. Rozkazy ALU z grup 3÷5 nie mogą być używane z grupą 2.

Tabela 17.9. Pole rozkazu rejestrowej ALU TI 8832 (pole 27)

Grupa 1		**Funkcja**
ADD	H#01	R + S + Cn
SUBR	H#02	(NOT R) + S + Cn
SUBS	H#03	R = (NOT S) + Cn
INSC	H#04	S + Cn
INCNS	H#05	(NOT S) + Cn
INCR	H#06	R + Cn
INCNR	H#07	(NOT R) + Cn
XOR	H#09	R XOR S
AND	H#0A	R AND S
OR	H#0B	R OR S
NAND	H#0C	R NAND S
NOR	H#0D	R NOR S
ANDNR	H#0E	(NOT R) AND S
Grupa 2		**Funkcja**
SRA	H#00	Przesunięcie arytmetyczne w prawo o pojedynczej precyzji
SRAD	H#10	Przesunięcie arytmetyczne w prawo o podwójnej precyzji
SRL	H#20	Przesunięcie logiczne w prawo o pojedynczej precyzji
SRLD	H#30	Przesunięcie logiczne w prawo o podwójnej precyzji
SLA	H#40	Przesunięcie arytmetyczne w lewo o pojedynczej precyzji
SLAD	H#50	Przesunięcie arytmetyczne w lewo o podwójnej precyzji
SLC	H#60	Przesunięcie cykliczne w lewo o pojedynczej precyzji

Tabela 17.9 (cd.)

Grupa 2		Funkcja
SLCD	H#70	Przesunięcie cykliczne w lewo o podwójnej precyzji
SRC	H#80	Przesunięcie cykliczne w prawo o pojedynczej precyzji
SRCD	H#90	Przesunięcie cykliczne w prawo o podwójnej precyzji
MQSRA	H#A0	Arytmetyczne przesunięcie w prawo rejestru MQ
MQSRL	H#B0	Logiczne przesunięcie w prawo rejestru MQ
MQSLL	H#C0	Logiczne przesunięcie w lewo rejestru MQ
MQSLC	H#D0	Cykliczne przesunięcie w lewo rejestru MQ
LOADMQ	H#E0	Ładuj rejestr MQ
PASS	H#F0	Przepuść ALU do Y (bez operacji przesuwania)
Grupa 3		**Funkcja**
SET1	H#08	Ustaw bit 1
SET0	H#18	Ustaw bit 0
TB1	H#28	Zbadaj bit 1
TB0	H#38	Zbadaj bit 0
ABS	H#48	Wartość bezwzględna
SMTC	H#58	Znak – moduł / uzupełnienie do dwóch
ADDI	H#68	Dodaj natychmiastowy
SUBI	H#78	Odejmij natychmiastowy
BADD	H#88	Bajtowo dodaj R do S
BSUBS	H#98	Bajtowo odejmij S od R
BSUBR	H#A8	Bajtowo odejmij R od S
BINCS	H#B8	Bajtowo inkrementuj S
BINCNS	H#C8	Bajtowo inkrementuj ujemną S
BXOR	H#D8	Bajtowo wykonaj XOR na R i S
BAND	H#E8	Bajtowo wykonaj AND na R i S
BOR	H#F8	Bajtowo wykonaj OR na R i S
Grupa 4		**Funkcja**
CRC	H#00	Akumuluj cykliczny znak redundancji
SEL	H#10	Wybierz S lub R
SNORM	H#20	Normalizuj pojedynczą długość
DNORM	H#30	Normalizuj podwójną długość
DIVRF	H#40	Ustal resztę dzielenia
SDIVQF	H#50	Ustal resztę dzielenia ze znakiem
SMULI	H#60	Iteruj mnożenie ze znakiem
SMULT	H#70	Zakończ mnożenie ze znakiem
SDIVIN	H#80	Inicjuj dzielenie ze znakiem
SDIVIS	H#90	Rozpocznij dzielenie ze znakiem
SDIVI	H#A0	Iteruj dzielenie ze znakiem
UDIVIS	H#B0	Rozpocznij dzielenie bez znaku
UDIVI	H#C0	Iteruj dzielenie bez znaku

Tabela 17.9. Pole rozkazu rejestrowej ALU TI 8832 (pole 27) (cd.)

Grupa 4		Funkcja
UMULI	H#D0	Iteruj mnożenie bez znaku
SDIVIT	H#E0	Zakończ dzielenie ze znakiem
UDIVIT	H#F0	Zakończ dzielenie bez znaku
Grupa 5		**Funkcja**
LOADFF	H#0F	Ładuj przerzutniki dzielenie/BCD
CLR	H#1F	Wyczyść
DUMPFF	H#5F	Wyjście z przerzutników dzielenie/BCD
BCDBIN	H#7F	Konwersja notacji BCD na binarną
EX3BC	H#8F	Korekta bajtu za pomocą kodu z nadmiarem 3
EX3C	H#9F	Korekta słowa za pomocą kodu z nadmiarem 3
SDIVO	H#AF	Badanie przepełnienia dzielenia ze znakiem
BINEX3	H#DF	Konwersja notacji binarnej na kod z nadmiarem 3
NOP32	H#FF	Brak operacji

17.5. Zastosowania mikroprogramowania

Od czasu wprowadzenia mikroprogramowania, a zwłaszcza od późnych lat sześćdziesiątych, zastosowania mikroprogramowania różnicowały się i upowszechniały. Już w roku 1971 można było odnotować większość (jeżeli nie wszystkie) współczesnych zastosowań mikroprogramowania [FLYN71]. Nowsze przeglądy obejmują w istocie ten sam zbiór zastosowań [RAUS80]. Do zbioru aktualnych zastosowań mikroprogramowania należą:

- budowa komputerów;
- emulacja;
- wspieranie systemów operacyjnych;
- budowa urządzeń o specjalnym przeznaczeniu;
- wspieranie języków wysokiego poziomu;
- mikrodiagnostyka;
- dostosowywanie do potrzeb użytkownika.

Rozdział ten był poświęcony dyskusji na temat budowy komputerów. Rozwiązanie mikroprogramowane jest jedną z metod wdrażania jednostek sterujących. Spokrewnioną metodą jest *emulacja* [MALL75]. Emulacja odnosi się do użycia mikroprogramu w jednej maszynie do wykonywania programów napisanych pierwotnie dla innej. Najbardziej powszechnym zastosowaniem emulacji jest wspomaganie użytkowników przenoszących się od komputera do komputera. W ten sposób postępują często producenci, aby ułatwić swoim klientom wymianę starszych maszyn na nowsze, dzięki czemu szukanie innego producenta (dostawcy) jest dla nich nieatrakcyjne. Użytkownicy są często zaskoczeni, stwierdzając, jak długo to przejściowe narzędzie jest stosowane. Jeden z obserwatorów [MALL83] zauważył, że nawet

w roku 1983 można było znaleźć System/370 IBM emulujący IBM 1401 zastąpiony fizycznie ponad 15 lat wcześniej!

Innym owocnym zastosowaniem mikroprogramowania jest *wspieranie systemów operacyjnych*. Mikroprogramy mogą być używane do implementacji funkcji pierwotnych, co zastępuje ważne części oprogramowania systemowego. Metoda ta może uprościć zadanie implementacji systemu operacyjnego i poprawić wydajność tego systemu.

Mikroprogramowanie jest również użytecznym narzędziem przy realizacji *urządzeń o specjalnym przeznaczeniu*, które mogą być wbudowane do komputera macierzystego. Dobrym tego przykładem jest moduł komunikacji danych (*data communications board*). Moduł ten zawiera własny mikroprocesor. Ponieważ jest używany do celów specjalnych, rozsądne jest wdrożenie niektórych jego funkcji w postaci oprogramowania układowego (*firmware*) zamiast zwykłego (*software*), co pozwala na zwiększenie wydajności.

Obsługa języków wysokiego poziomu jest jeszcze jednym owocnym obszarem zastosowań mikroprogramowania. Różne funkcje i rodzaje danych mogą być zaimplementowane bezpośrednio w postaci oprogramowania układowego. W rezultacie łatwiej jest kompilować program na efektywną postać języka maszynowego. Dzięki temu język maszynowy jest dostosowywany do potrzeb języka wysokiego poziomu (jak FORTRAN, COBOL, Ada).

Mikroprogramowanie może być używane do wspierania monitorowania, wykrywania, izolowania i naprawiania błędów systemu. Możliwości te są znane jako *mikrodiagnostyka* i mogą znacznie ułatwić konserwację systemu. Rozwiązanie to umożliwia automatyczną rekonfigurację systemu po wykryciu uszkodzenia: jeśli na przykład niewłaściwie pracuje szybki układ mnożący, jego funkcję może przejąć układ mikroprogramowany.

Bardzo ogólną kategorią zastosowań jest *dostosowywanie do potrzeb użytkownika*. W wielu maszynach występuje *zapisywalna pamięć sterowania*, to znaczy pamięć sterowania zaimplementowana w postaci RAM zamiast ROM, co umożliwia użytkownikowi pisanie mikroprogramów. Na ogół dostarczana jest łatwa do użycia lista mikrorozkazów o formacie pionowym. Dzięki temu użytkownik może dostosować maszynę do określonego zastosowania.

17.6. Polecana literatura

Istnieje wiele książek poświęconych mikroprogramowaniu. Być może najbardziej obszerną jest [LYNC93]. W [SEGE91] przedstawiono podstawy mikroprogramowania i projektowania systemów mikroprogramowanych, posługując się przykładowym projektem prostego procesora 16-bitowego. Również w [CART96] przedstawiono podstawowe koncepcje, posługując się próbnym komputerem. W [PARK89] i [TI90] zawarto szczegółowy opis modułu rozwijania oprogramowania TI 8800.

CART96 Carter J.: *Microprocessor Architecture and Microprogramming.* Upper Saddle River, Prentice Hall, 1996.

LYNC93 Lynch M.: *Microprogrammed State Machine Design*. Boca Raton, 1993.

PARK89 Parker A., Hamblen J.: *An Introduction to Microprogramming with Exercises Designed for Texas Instruments SN74ACT8800 Software Development Board*. Dallas, Texas Instruments, 1989.

SEGE91 Segee B., Field J.: *Microprogramming and Computer Architecture*. New York, Wiley, 1991.

TI90 Texas Instruments Inc.: *SN74ACT880 Family Data Manual*. SCSS006C, 1990.

17.7. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Język mikroprogramowania – *microprogramming language*
Kodowanie mikrorozkazu – *microinstruction encoding*
Mikroprogram – *microprogram*
Mikroprogramowana jednostka sterująca – *microprogrammed control unit*
Mikroprogramowanie programowe – *soft microprogramming*
Mikroprogramowanie układowe – *hard microprogramming*
Mikrorozkaz nieupakowany – *unpacked microinstruction*
Mikrorozkaz pionowy – *vertical microinstruction*
Mikrorozkaz poziomy – *horizontal microinstruction*
Mikrorozkazy – *microinstructions*
Oprogramowanie układowe – *firmware*
Pamięć sterująca – *control memory*
Słowo sterujące – *control word*
Szeregowanie mikrorozkazów – *microinstruction sequencing*
Wykonywanie mikrorozkazu – *microinstruction execution*

Pytania kontrolne

17.1. Jaka jest różnica między układową a mikroprogramowaną implementacją jednostki sterującej?
17.2. Jak interpretuje się mikrorozkaz poziomy?
17.3. Jaka jest rola pamięci sterowania?
17.4. Jaka jest typowa sekwencja wykonywania mikrorozkazu poziomego?
17.5. Jaka jest różnica między mikrorozkazem poziomym a pionowym?
17.6. Jakie są podstawowe zadania realizowane przez mikroprogramowaną jednostkę sterującą?
17.7. Jaka jest różnica między mikrorozkazem upakowanym a nieupakowanym?
17.8. Jaka jest różnica między mikroprogramowaniem układowym a programowym?
17.9. Jaka jest różnica między kodowaniem funkcjonalnym a kodowaniem według zasobów?
17.10. Wymień powszechne zastosowania mikroprogramowania.

Problemy do rozwiązania

17.1. Opisz implementację rozkazu mnożenia w hipotetycznej maszynie zaprojektowanej przez Wilkesa. Użyj opisu oraz sieci działań.

17.2. Załóż, że lista mikrorozkazów zawiera mikrorozkaz o następującej postaci symbolicznej:

```
IF(AC₀ = 1) THEN CAR ← (C₀₋₆) ELSE CAR ← (CAR) + 1
```

gdzie AC_0 jest bitem znaku akumulatora, a $C_{0\text{-}6}$ to pierwsze 7 bitów mikrorozkazu. Stosując ten mikrorozkaz, napisz mikroprogram wdrażający rozkaz maszynowy Branch Register Minus (BRM), który powoduje rozgałęzienie, jeśli AC jest ujemny. Załóż, że bity $C_1 \div C_n$ mikrorozkazu określają równoległy zbiór mikrooperacji. Wyraź program symbolicznie.

17.3. Prosty procesor ma cztery główne fazy w cyklu rozkazu: pobierania, adresowania pośredniego, wykonywania i przerwania. Dwa 1-bitowe znaczniki czasu określają bieżącą fazę w odniesieniu do realizacji układowej.

(a) Dlaczego potrzebne są te znaczniki stanu?

(b) Dlaczego nie są one potrzebne w przypadku mikroprogramowanej jednostki sterującej?

17.4. Rozważ jednostkę sterującą z rys. 17.7. Załóż, że pamięć sterowania ma szerokość 24 bity. Część sterowania w formacie mikrorozkazu jest podzielona na 2 pola. Pole mikrooperacji złożone z 13 bitów określa mikrooperację, która ma być zrealizowana. Pole wyboru adresu określa warunek, oparty na znacznikach stanu, który powoduje rozgałęzienie mikrorozkazu. Znaczników stanu jest 8.

(a) Ile bitów jest w polu wyboru adresu?

(b) Ile bitów jest w polu adresu?

(c) Jaki jest rozmiar pamięci sterowania?

17.5. Jak w warunkach z poprzedniego problemu może być realizowane rozgałęzienie bezwarunkowe? Jak można zapobiec rozgałęzieniu – to znaczy, opisz mikrorozkaz, który nie określa rozgałęziania ani warunkowego, ani bezwarunkowego.

17.6. Chcemy, żeby każdemu podprogramowi rozkazu maszynowego odpowiadało 8 słów sterowania. Kody operacji rozkazów maszynowych mają 5 bitów, a pamięć sterowania ma pojemność 1024 słów. Zaproponuj odwzorowanie rejestru rozkazu w rejestrze adresu sterowania.

17.7. Używany jest kodowany format mikrorozkazu. Pokaż, w jaki sposób 9-bitowe pole mikrooperacji może być podzielone na pola cząstkowe w celu określenia 46 różnych działań.

17.8. Procesor ma 16 rejestrów, ALU o 16 funkcjach logicznych i 16 arytmetycznych oraz przesuwnik o 8 operacjach, przy czym zespoły te są połączone za pomocą wewnętrznej magistrali. Zaprojektuj format mikrorozkazu określający różne mikrooperacje tego procesora.

Część piąta
ORGANIZACJA RÓWNOLEGŁA

Problemy przedstawione w części piątej

Końcowa część książki została poświęcona coraz ważniejszej dziedzinie organizacji równoległej. W tego rodzaju organizacji współdziała ze sobą wiele jednostek przetwarzania, wykonując programy aplikacyjne. Podczas gdy w procesorze superskalarnym wykorzystuje się możliwość równoległej pracy na poziomie rozkazów, w organizacji przetwarzania równoległego poszukuje się pełniejszego stopnia paralelizmu, umożliwiającego równoległe wykonywanie pracy przez wiele współdziałających ze sobą procesorów. Tego rodzaju organizacja rodzi szereg problemów. Jeśli na przykład wiele procesorów, z których każdy ma własną pamięć podręczną, ma dostęp do tej samej pamięci, musi być zastosowane rozwiązanie sprzętowe lub programowe zapewniające, że procesory te współużytkują ważny (aktualny) obraz pamięci głównej; zagadnienie to jest znane jako problem spójności pamięci podręcznej. Oprócz innych problemów, w części piątej zostały przedstawione zagadnienia projektowania.

Tematyka rozdziałów części piątej

Rozdział 18. Przetwarzanie równoległe

Rozdział 18 rozpoczyna się od przeglądu zagadnień przetwarzania równoległego. Następnie będą przedstawione trzy podejścia do organizacji wieloprocesorowych: wieloprocesory symetryczne (SMP), klastry i maszyny oparte na niejednorodnym dostępie do pamięci (NUMA). SMP i klastry to najczęściej stosowane rozwiązania organizacji wielu procesorów w celu zwiększenia wydajności i poprawy dostępności. Systemy NUMA to koncepcja nowsza, która nie osiągnęła jeszcze szerokiego powodzenia komercyjnego, lecz jest wielce obiecująca. Na zakończenie rozdziału 18 zostanie przedstawiona wyspecjalizowana organizacja nosząca nazwę procesora wektorowego.

Rozdział 18

Przetwarzanie równoległe

18.1. Organizacje wieloprocesorowe
Rodzaje systemów z procesorami równoległymi
Organizacje równoległe
18.2. Wieloprocesory symetryczne
Organizacja
Rozważania projektowe dotyczące wieloprocesorowych systemów operacyjnych
Duże komputery SMP
18.3. Spójność pamięci podręcznych i protokół MESI
Rozwiązania programowe
Rozwiązania sprzętowe
Protokół MESI
18.4. Klastry
Konfiguracja klastra
Problemy projektowania systemu operacyjnego
Architektura systemu klastrowego
Klastry a SMP
18.5. Niejednorodny dostęp do pamięci
Uzasadnienie
Organizacja
Za i przeciw rozwiązaniu NUMA
18.6. Obliczenia wektorowe
Sposoby rozwiązywania obliczeń wektorowych
Urządzenie wektorowe IBM 3090
18.7. Polecana literatura
18.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania
Podstawowe terminy i ich angielskie odpowiedniki
Pytania kontrolne
Problemy do rozwiązania

Podstawowe spostrzeżenia

- Tradycyjnym sposobem zwiększania wydajności systemu jest zastosowanie wielu procesorów, które mogą realizować daną pracę równolegle. Dwoma najczęściej stosowanymi rozwiązaniami organizacji wieloprocesorowych są wieloprocesory symetryczne (SMP) i klastry. Niedawno zostały wprowadzone na rynek systemy oparte na niejednorodnym dostępie do pamięci (NUMA).
- SMP składa się z wielu podobnych procesorów w ramach tego samego komputera, połączonych ze sobą magistralą lub za pośrednictwem jakiegoś układu przełączającego. W przypadku SMP najbardziej krytycznym problemem jest spójność pamięci podręcznych. Każdy procesor dysponuje własną pamięcią podręczną; jest więc możliwe, że określony wiersz danych jest obecny w wielu pamięciach podręcznych. Jeśli zostanie on zmieniony w jednej pamięci, to zarówno pamięć główna, jak i pozostałe pamięci podręczne miałyby nieważną wersję tego wiersza. W celu rozwiązania tego problemu powstają protokoły spójności pamięci podręcznych.
- Klaster jest grupą wzajemnie połączonych, kompletnych komputerów funkcjonujących razem jako jednolity system obliczeniowy, który może stwarzać iluzję, że jest jednym komputerem. Określenie *kompletny komputer* oznacza maszynę zdolną do samodzielnej pracy poza klastrem.
- System NUMA to wieloprocesory o wspólnie użytkowanej pamięci, w którym czas dostępu danego procesora do słowa w pamięci zależy od lokacji tego słowa.
- Rodzajem organizacji równoległej przeznaczonym do celów specjalnych jest urządzenie wektorowe, dostosowane do przetwarzania wektorów (lub tablic) danych.

Tradycyjnie komputer był postrzegany jako maszyna sekwencyjna. Większość komputerowych języków programowania wymaga od programisty określania algorytmów jako sekwencji rozkazów. Procesory realizują programy przez sekwencyjne wykonywanie rozkazów maszynowych, rozkaz po rozkazie. Każdy rozkaz jest wykonywany w postaci sekwencji operacji (pobieranie rozkazu, pobieranie argumentów, przeprowadzanie operacji, zapisanie wyników).

Taki obraz komputera nigdy nie był całkowicie prawdziwy. Na poziomie mikrooperacji wiele sygnałów sterujących jest generowanych w tym samym czasie. Potokowe przetwarzanie rozkazów, polegające przynajmniej na nakładaniu się operacji pobierania i wykonywania, jest stosowane od długiego czasu. Są to dwa przykłady równoległego realizowania funkcji. Rozwinięciem tego podejścia jest organizacja superskalarna, wykorzystująca paralelizm na poziomie rozkazu. W maszynie superskalarnej w pojedynczym procesorze występuje wiele jednostek wykonujących, które mogą równolegle wykonywać wiele rozkazów pochodzących z tego samego programu.

W miarę ewoluowania techniki komputerowej i spadku kosztu sprzętu komputerowego, projektanci komputerów dostrzegali coraz więcej możliwości paralelizmu, mając na uwadze zwykle podniesienie wydajności, a czasem poprawę niezawodności. W rozdziale tym zapoznamy się z trzema najwybitniejszymi i najbardziej udanymi podejściami do organizacji równoległej. Po dokonaniu przeglądu ogólnego w tym rozdziale zajmiemy się trzema najwybitniejszymi rozwiązaniami organizacji równoległej. Najpierw przeanalizujemy wieloprocesory symetryczne (SMP), jeden z najwcześniejszych i wciąż najbardziej rozpowszechniony przykład organizacji równoległej. W organizacji SMP wiele procesorów wspólnie użytkuje pamięć. Z taką organizacją wiąże się problem spójności pamięci podręcznych, któremu zostanie poświęcony osobny podrozdział. Następnie opiszemy klastry, które składają się z wielu niezależnych komputerów współdziałających ze sobą w sposób zorganizowany. Klastry coraz częściej służą do realizowania prac o wielkości wykraczającej poza potencjał jednego SMP. Trzecim rozwiązaniem zastosowania wielu procesorów, jakie przeanalizujemy, są systemy oparte na niejednorodnym dostępie do pamięci (NUMA). Jest to podejście stosunkowo nowe i nie zostało jeszcze wypróbowane na rynku, jednak często jest rozważane jako rozwiązanie alternatywne w stosunku do SMP i klastrów. Na zakończenie zapoznamy się ze sprzętowymi rozwiązaniami organizacyjnymi służącymi do obliczeń wektorowych. W tych rozwiązaniach optymalizuje się ALU pod kątem przetwarzania wektorów lub tablic liczb zmiennopozycyjnych. Są one powszechnie spotykane w klasie systemów znanej jako *superkomputery*.

18.1. Organizacje wieloprocesorowe

Rodzaje systemów z procesorami równoległymi

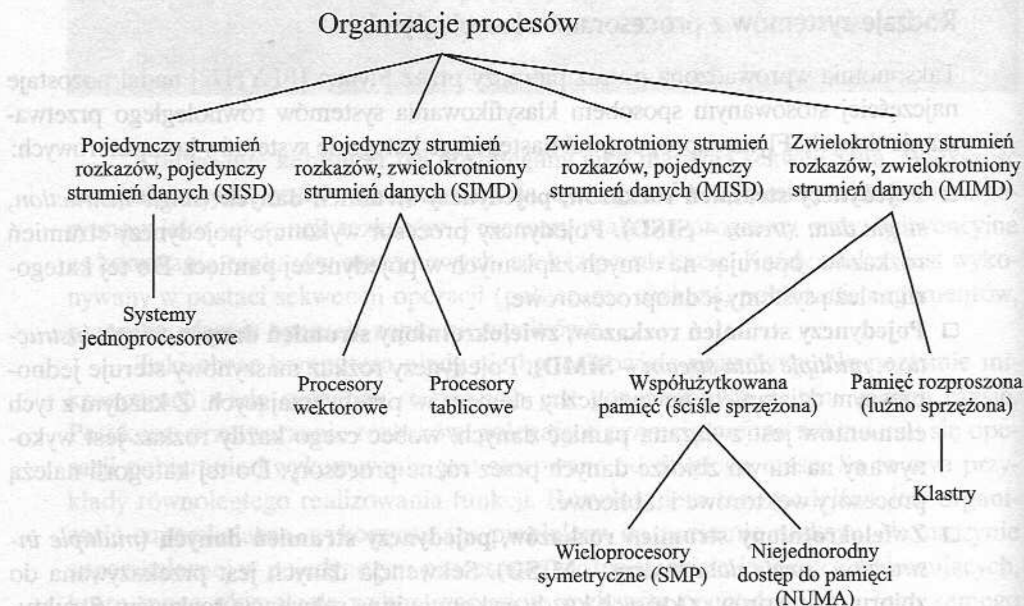
Taksonomia wprowadzona po raz pierwszy przez Flynna [FLYN72] nadal pozostaje najczęściej stosowanym sposobem klasyfikowania systemów równoległego przetwarzania danych. Flynn zaproponował następujące kategorie systemów komputerowych:

- **Pojedynczy strumień rozkazów, pojedynczy strumień danych** (*single instruction, single data stream* – **SISD**). Pojedynczy procesor wykonuje pojedynczy strumień rozkazów, operując na danych zapisanych w pojedynczej pamięci. Do tej kategorii należą systemy jednoprocesorowe.
- **Pojedynczy strumień rozkazów, zwielokrotniony strumień danych** (*single instruction, multiple data stream* – **SIMD**). Pojedynczy rozkaz maszynowy steruje jednoczesnym działaniem pewnej liczby elementów przetwarzających. Z każdym z tych elementów jest związana pamięć danych, wobec czego każdy rozkaz jest wykonywany na innym zbiorze danych przez różne procesory. Do tej kategorii należą procesory wektorowe i tablicowe.
- **Zwielokrotniony strumień rozkazów, pojedynczy strumień danych** (*multiple instruction, single data stream* – **MISD**). Sekwencja danych jest przekazywana do zbioru procesorów, z których każdy wykonuje inną sekwencję rozkazów. Struktura taka nie została implementowana komercyjnie.

- **Zwielokrotniony strumień rozkazów, zwielokrotniony strumień danych** (*multiple instruction, multiple data stream* – **MIMD**). Zbiór procesorów jednocześnie wykonuje różne sekwencje rozkazów na różnych zbiorach danych. Do tej kategorii należą systemy SMP, klastry i NUMA.

W organizacji MIMD są stosowane procesory ogólnego przeznaczenia; każdy z nich jest zdolny do przetwarzania wszelkich rozkazów koniecznych do dokonania pożądanej transformacji danych. MIMD mogą być dalej podzielone zależnie od środków wzajemnego komunikowania się procesorów (rys. 18.1). Jeśli procesory wspólnie użytkują jedną pamięć, to każdy z nich sięga do zapisanych w niej programów i danych i komunikują się za pośrednictwem tej pamięci. Najczęściej spotykaną postacią takiego systemu jest **wieloprocesor symetryczny** (*symmetric multiprocessor* – **SMP**), który przeanalizujemy w podrozdz. 18.2. W SMP wiele procesorów współużytkuje jedną pamięć lub zbiór pamięci, posługując się wspólną magistralą lub innym mechanizmem połączeń; cechą odróżniającą takie rozwiązania jest to, że czas dostępu do dowolnego rejonu pamięci jest w przybliżeniu taki sam dla każdego procesora. Nowszym rozwiązaniem jest organizacja o **niejednorodnym dostępie do pamięci** (*nonuniform memory access* – **NUMA**), która zostanie opisana w podrozdz. 18.5. Jak sugeruje nazwa, czas dostępu do różnych rejonów pamięci może być różny dla poszczególnych procesorów NUMA.

Zbiór niezależnych komputerów jednoprocesorowych lub SMP może być wzajemnie połączony w celu utworzenia **klastra**. Komunikacja między takimi komputerami jest realizowana bądź za pośrednictwem stałych ścieżek, bądź pewnego rodzaju rozwiązania sieciowego.

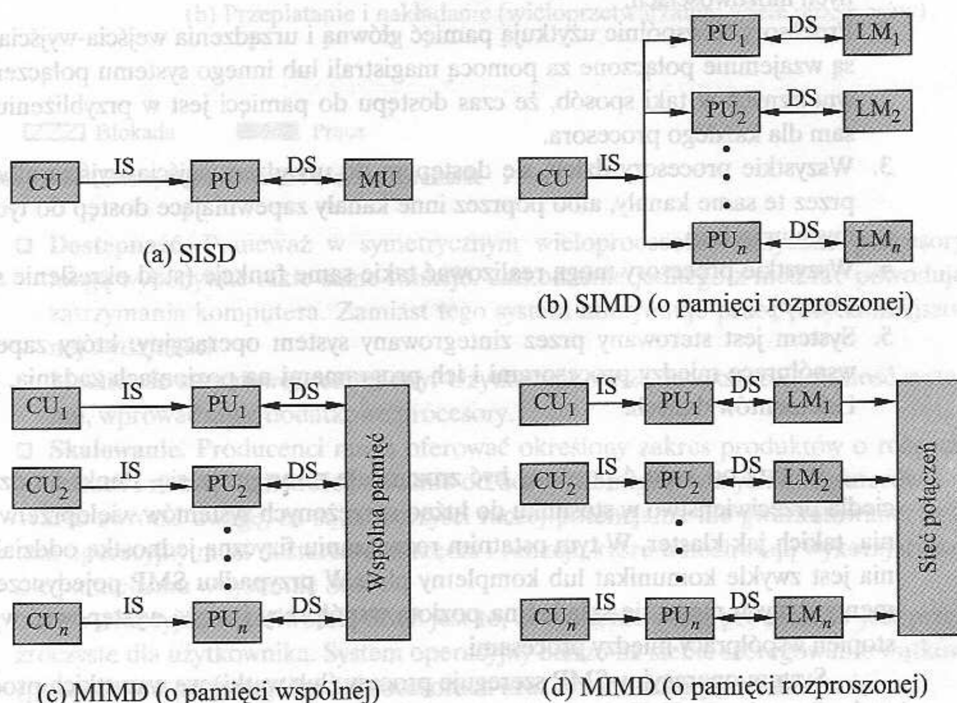


Rysunek 18.1. Taksonomia architektur równoległych procesorów

Organizacje równoległe

Na rysunku 18.2 została przedstawiona ogólna organizacja wybranych systemów z rys. 18.1. Na rysunku 18.2a pokazano strukturę SISD. Istnieje tutaj pewien rodzaj jednostki sterującej (CU) doprowadzającej strumień rozkazów (IS) do jednostki przetwarzającej (PU). Jednostka ta operuje na pojedynczym strumieniu danych (DS) z jednostki pamięciowej (MU). W przypadku SIMD nadal występuje pojedyncza jednostka sterowania, która w tym przypadku kieruje pojedynczy strumień rozkazów do wielu jednostek przetwarzających. Każda z PU może mieć własną, wyspecjalizowaną pamięć (pokazaną na rys. 18.2b) lub pamięć może być wspólna. Wreszcie w przypadku MIMD występuje wiele jednostek sterujących, z których każda kieruje oddzielny strumień rozkazów do własnej jednostki przetwarzającej. MIMD może być wieloprocesorem o wspólnej pamięci (rys. 18.2c) lub wielokomputerem o pamięci rozproszonej (rys. 18.2d).

Zagadnienia projektowe związane z SMP, klastrami i systemami NUMA są złożone i dotyczą organizacji fizycznej, struktury wzajemnych połączeń, komunikacji



Rysunek 18.2. Alternatywne organizacje komputerów

między procesorami, projektowania systemu operacyjnego i rozwiązań oprogramowania aplikacyjnego. Tutaj interesuje nas głównie organizacja, chociaż poruszymy krótko również problemy projektowania systemów operacyjnych.

18.2. Wieloprocesory symetryczne

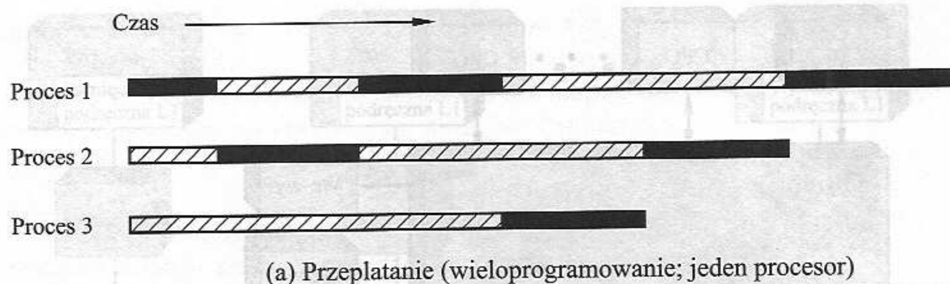
Aż do niedawna praktycznie wszystkie komputery osobiste i większość stacji roboczych zawierała pojedynczy mikroprocesor ogólnego przeznaczenia. W miarę jak wzrastało zapotrzebowanie na wzrost wydajności i spadała cena mikroprocesorów, producenci zaczęli wprowadzać systemy o organizacji SMP. Określenie SMP odnosi się do sprzętowej architektury komputera oraz do właściwości systemu operacyjnego, które odzwierciedlają tę architekturę. SMP można zdefiniować jako samodzielny system komputerowy o następujących właściwościach:

1. Występują w nich dwa lub większa liczba podobnych procesorów o porównywalnych możliwościach.
2. Procesory te wspólnie użytkują pamięć główną i urządzenia wejścia-wyjścia oraz są wzajemnie połączone za pomocą magistrali lub innego systemu połączeń wewnętrznych w taki sposób, że czas dostępu do pamięci jest w przybliżeniu taki sam dla każdego procesora.
3. Wszystkie procesory dzielą się dostępem do urządzeń wejścia-wyjścia albo poprzez te same kanały, albo poprzez inne kanały zapewniające dostęp do tych samych urządzeń.
4. Wszystkie procesory mogą realizować takie same funkcje (stąd określenie *symetryczne*).
5. System jest sterowany przez zintegrowany system operacyjny, który zapewnia współpracę między procesorami i ich programami na poziomach zadania, pliku i elementów danych.

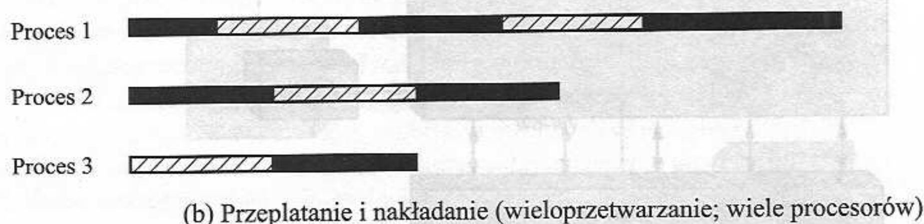
Punkty od 1 do 4 powinny być zrozumiałe same przez się. Punkt 5 odzwierciedla przeciwieństwo w stosunku do luźno sprzężonych systemów wieloprzetwarzania, takich jak klaster. W tym ostatnim rozwiązaniu fizyczną jednostką oddziaływania jest zwykle komunikat lub kompletny plik. W przypadku SMP pojedyncze elementy danych mogą się składać na poziom współpracy i może występować wysoki stopień współpracy między procesami.

System operacyjny SMP szereguje procesy (lub wątki) we wszystkich procesorach. Organizacja SMP ma wiele potencjalnych przewag nad organizacjami jednoprocesorowymi; należą do nich:

- **Wydajność.** Jeśli praca, jaką ma wykonać komputer, może być zorganizowana w taki sposób, że określone jej części mogą być wykonywane równolegle, to system z wieloma procesorami pozwoli uzyskać większą wydajność niż z jednym procesorem takiego samego rodzaju (rys. 18.3).



(a) Przeplatanie (wieloprogramowanie; jeden procesor)



(b) Przeplatanie i nakładanie (wieloprzetwarzanie; wiele procesorów)

Blokada   Praca

Rysunek 18.3. Wieloprogramowanie i wieloprzetwarzanie

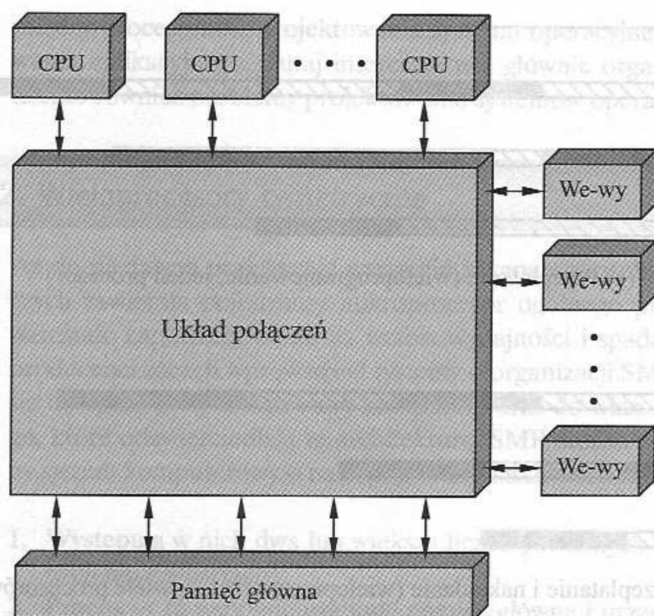
- **Dostępność.** Ponieważ w symetrycznym wieloprocesorze wszystkie procesory mogą wykonywać takie same funkcje, uszkodzenie jednego z nich nie powoduje zatrzymania komputera. Zamiast tego system kontynuuje pracę przy zmniejszonej wydajności.
- **Możliwość stopniowej rozbudowy.** Użytkownik może zwiększać wydajność systemu, wprowadzając dodatkowe procesory.
- **Skalowanie.** Producenci mogą oferować określony zakres produktów o różnych cenach i różnej wydajności, zależnie od liczby skonfigurowanych komputerów.

Warto zwrócić uwagę, że są to korzyści raczej potencjalne niż gwarantowane. System operacyjny musi dostarczyć narzędzi i funkcji, które umożliwiają wykorzystywanie paralelizmu w systemie SMP.

Atrakcyjną właściwością SMP jest to, że istnienie wielu procesorów jest przezroczyste dla użytkownika. System operacyjny bierze na siebie szeregowanie wątków (procesów) w poszczególnych procesorach oraz synchronizację ich działania.

Organizacja

Na rysunku 18.4 jest pokazany ogólny schemat organizacji systemu wieloprocesorowego. Występuje w nim dwa lub więcej procesorów. Każdy procesor stanowi zamkniętą całość, obejmując jednostkę sterującą, ALU, rejestry i zwykle jeden lub dwa poziomy pamięci podręcznej. Każdy procesor ma dostęp do wspólnej pamięci



Rysunek 18.4. Ogólny schemat blokowy silnie powiązanego systemu mikroprocesorowego

głównej oraz do urządzeń wejścia-wyjścia za pośrednictwem pewnej formy mechanizmu łączenia. Procesory mogą porozumiewać się ze sobą poprzez pamięć (wiadomości i informacje o stanie pozostawiane we wspólnych obszarach danych). Mogą również wymieniać sygnały bezpośrednio. Pamięć jest często zorganizowana w ten sposób, że możliwych jest wiele jednoczesnych dostępów do oddzielnych bloków pamięci. W niektórych konfiguracjach każdy procesor może dysponować własną pamięcią główną i kanałami wejścia-wyjścia jako uzupełnieniem zasobów wspólnych.

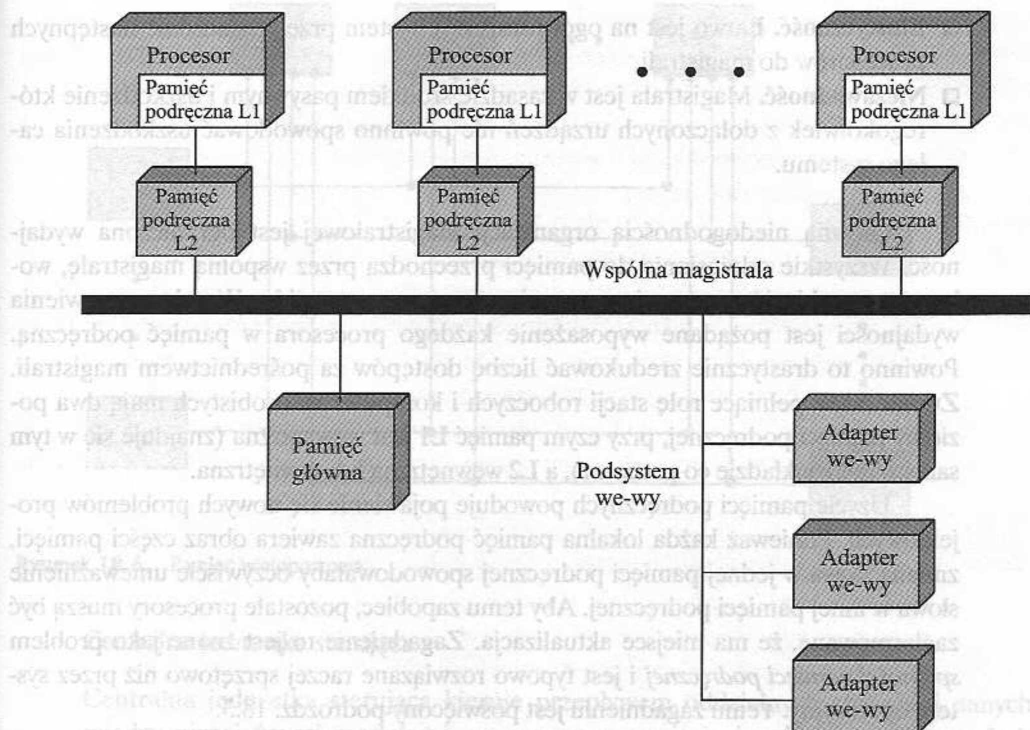
Organizacje systemów wieloprocesorowych można sklasyfikować jako systemy z:

- magistralą z podziałem czasu lub ze wspólną magistralą;
- pamięcią wieloportową;
- centralną jednostką sterującą.

Magistrala z podziałem czasu

Magistrala z podziałem czasu jest najprostszym mechanizmem umożliwiającym zbudowanie systemu wieloprocesorowego (rys. 18.5). Struktura i interfejsy są w zasadzie takie same, jak w przypadku systemu jednoprocesorowego używającego połączenia magistralowego. Magistrala składa się z linii sterowania, linii adresu i danych. Aby ułatwić transfery DMA* z procesorów wejścia-wyjścia, są przewidziane następujące rozwiązania:

* DMA oznacza dostęp bezpośredni do pamięci (*Direct Memory Access*) (przyp. tłum.).



Rysunek 18.5. Organizacja wieloprocesora symetrycznego

- **Adresowanie.** Musi istnieć możliwość rozróżniania modułów na magistrali w celu określenia źródła i miejsca docelowego danych.
- **Arbitraż.** Dowolny moduł wejścia-wyjścia może okresowo funkcjonować jako jednostka nadrzędna. Przewidziano mechanizm arbitrażu między jednostkami rywalizującymi o sterowanie magistralą, korzystający z pewnego rodzaju schematu priorytetów.
- **Podział czasu.** Gdy jeden z modułów steruje magistralą, pozostałe są odłączone i muszą w razie konieczności zawiesić działanie, aż do uzyskania dostępu do magistrali.

Te jednoprocesorowe rozwiązania są bezpośrednio użyteczne w konfiguracji wieloprocesorowej. W tym przypadku istnieje wiele jednostek centralnych i wiele procesorów wejścia-wyjścia, z których wszystkie dążą do uzyskania dostępu do jednego lub wielu modułów pamięci poprzez magistralę.

Organizacja magistralowa ma wiele zalet w porównaniu z innymi rozwiązaniami.

- **Prostota.** Jest to najprostsze rozwiązanie organizacji wieloprocesorowej. Interfejs fizyczny, adresowanie, arbitraż i układy logiczne podziału czasu w każdym procesorze pozostają takie same jak w systemie jednoprocesorowym.

- **Elastyczność.** Łatwo jest na ogół rozszerzyć system przez dołączenie następnych procesorów do magistrali.
- **Niezawodność.** Magistrala jest w zasadzie środkiem pasywnym i uszkodzenie któregokolwiek z dołączonych urządzeń nie powinno spowodować uszkodzenia całego systemu.

Główną niedogodnością organizacji magistralowej jest ograniczona wydajność. Wszystkie odniesienia do pamięci przechodzą przez wspólną magistralę, wobec tego szybkość systemu jest ograniczana przez czas cyklu. W celu poprawienia wydajności jest pożądane wyposażenie każdego procesora w pamięć podręczną. Powinno to drastycznie zredukować liczbę dostępów za pośrednictwem magistrali. Zwykle SMP pełniące rolę stacji roboczych i komputerów osobistych mają dwa poziomy pamięci podręcznej, przy czym pamięć L1 jest wewnętrzna (znajduje się w tym samym mikroukładzie co procesor), a L2 wewnętrzna lub zewnętrzna.

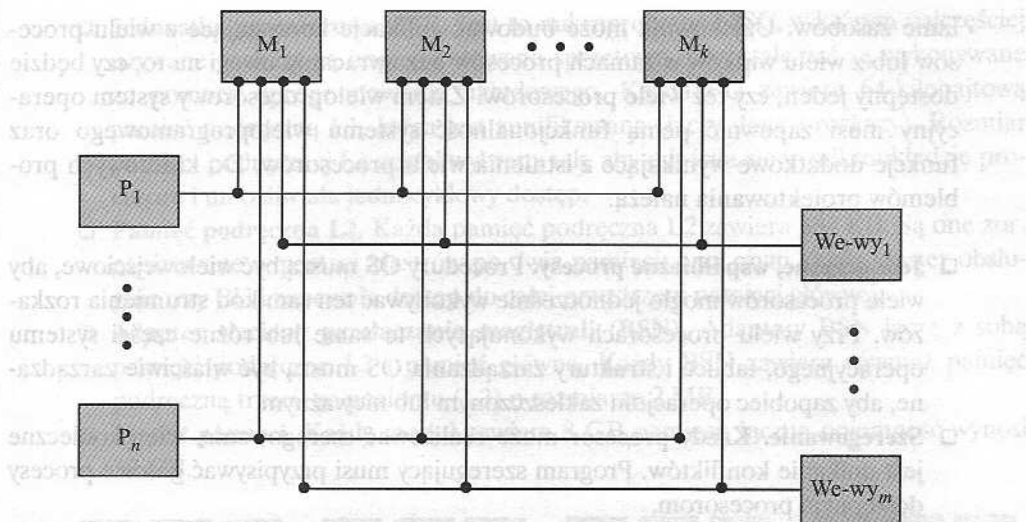
Użycie pamięci podręcznych powoduje pojawienie się nowych problemów projektowych. Ponieważ każda lokalna pamięć podręczna zawiera obraz części pamięci, zmiana słowa w jednej pamięci podręcznej spowodowałaby oczywiście unieważnienie słowa w innej pamięci podręcznej. Aby temu zapobiec, pozostałe procesory muszą być zaalarmowane, że ma miejsce aktualizacja. Zagadnienie to jest znane jako problem *spójności pamięci podręcznej* i jest typowo rozwiązane raczej sprzętowo niż przez system operacyjny. Temu zagadnieniu jest poświęcony podrozdz. 18.3.

Pamięć wieloportowa

Użycie pamięci wieloportowej umożliwia bezpośredni, niezależny dostęp każdego procesora i każdego modułu wejścia-wyjścia do modułów pamięci głównej (rys. 18.6). Do rozwiązywania konfliktów są wymagane układy logiczne związane z pamięcią. Często stosowaną metodą rozwiązywania konfliktów jest stałe przypisanie priorytetu każdemu portowi pamięci. Zwykle interfejs fizyczny i elektryczny każdego portu jest identyczny z tym, jaki występowałby w jednoportowym module pamięci. Potrzeba więc niewiele (lub wcale) modyfikacji procesora lub modułów wejścia-wyjścia, aby dostosować je do pamięci wieloportowej.

Rozwiązanie z pamięcią wieloportową jest bardziej złożone niż magistralowe, wymaga bowiem dodania do systemu pamięci dość dużej liczby układów logicznych. Powinno jednak umożliwić uzyskanie większej wydajności, ponieważ każdy procesor ma własną ścieżkę do każdego modułu pamięci. Inną zaletą rozwiązania wieloportowego jest umożliwienie skonfigurowania części pamięci jako pamięci „własnych" jednego lub wielu procesorów i (lub) modułów wejścia-wyjścia. Właściwość ta pozwala na poprawę zabezpieczenia przed nieupoważnionym dostępem oraz na przechowywanie podprogramów regeneracji w obszarach pamięci, które nie mogą być modyfikowane przez inne procesory.

Zwróćmy uwagę na to, że do sterowania pamięciami podręcznymi powinna być używana metoda jednoczesnego zapisu, ponieważ nie ma innych środków alarmowania innych procesorów o aktualizacji pamięci.



Rysunek 18.6. Pamięć wieloportowa

Centralna jednostka sterująca

Centralna jednostka sterująca kieruje przepływem oddzielnych strumieni danych między niezależnymi modułami: procesorem, pamięcią i wejściem-wyjściem. Jednostka ta może buforować zapotrzebowania, dokonywać arbitrażu i realizować taktowanie. Może również przekazywać wiadomości dotyczące stanu i sterowania między procesorami, a także zawiadamiać o aktualizacji pamięci podręcznej.

Ponieważ wszystkie układy logiczne związane z koordynowaniem konfiguracji wieloprocesorowej znajdują się w centralnej jednostce sterującej, interfejsy wejścia-wyjścia i pamięci procesora pozostają w zasadzie niezmienione. Umożliwia to taką samą elastyczność i prostotę interfejsu, jak w przypadku rozwiązania magistralowego. Główną wadą tego rozwiązania jest złożoność jednostki sterującej, która jest potencjalnie wąskim gardłem wydajności.

Struktura oparta na centralnej jednostce sterującej była w swoim czasie stosunkowo rozpowszechniona w dużych systemach wieloprocesorowych, takich jak duże modele z rodziny S/370 IBM. Obecnie należy do rzadkości.

Rozważania projektowe dotyczące wieloprocesorowych systemów operacyjnych

System operacyjny SMP tak zarządza procesorem i innymi zasobami komputera, że użytkownik dostrzega tylko pojedynczy system operacyjny sterujący zasobami systemu. W istocie tego rodzaju konfiguracja powinna być postrzegana jako jednoprocesorowy system wieloprogramowy. Zarówno w przypadku SMP, jak i systemu jednoprocesorowego, jednocześnie może być czynnych wiele zadań lub procesów, rolą zaś systemu operacyjnego jest szeregowanie ich wykonywania oraz przydzie-

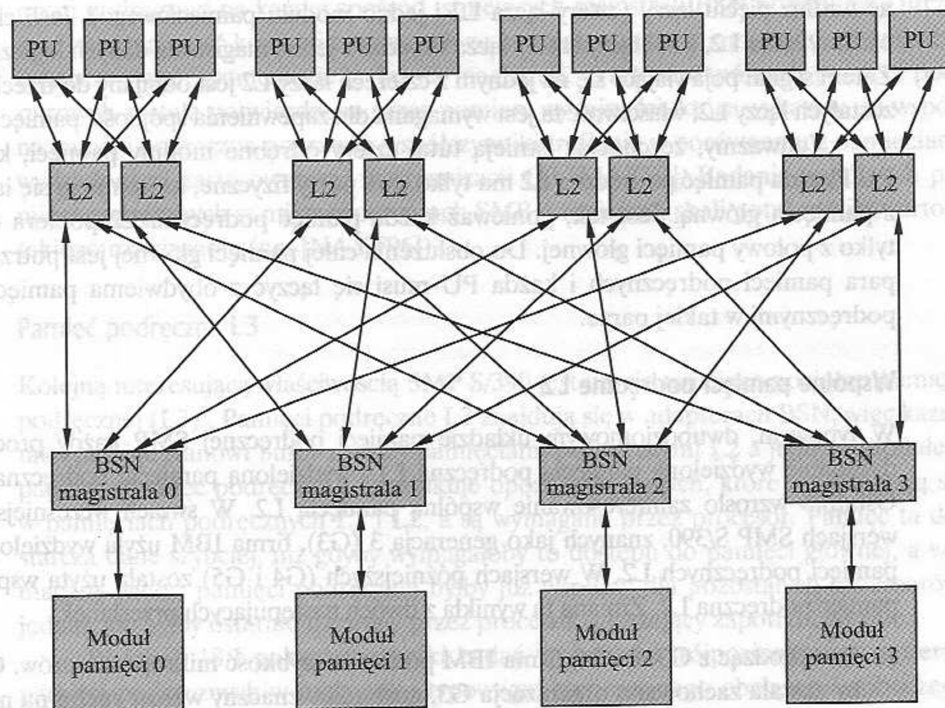
lanie zasobów. Użytkownik może budować aplikacje korzystające z wielu procesów lub z wielu wątków w ramach procesów bez zwracania uwagi na to, czy będzie dostępny jeden, czy też wiele procesorów. Zatem wieloprocesorowy system operacyjny musi zapewnić pełną funkcjonalność systemu wieloprogramowego oraz funkcje dodatkowe wynikające z istnienia wielu procesorów. Do kluczowych problemów projektowania należą:

- **Jednoczesne, współbieżne procesy.** Procedury OS muszą być wielowejściowe, aby wiele procesorów mogło jednocześnie wykonywać ten sam kod strumienia rozkazów. Przy wielu procesorach wykonujących te same lub różne części systemu operacyjnego, tablice i struktury zarządzania OS muszą być właściwie zarządzane, aby zapobiec operacjom zakleszczonym lub nieważnym.
- **Szeregowanie.** Każdy procesor może realizować szeregowanie, więc konieczne jest unikanie konfliktów. Program szeregujący musi przypisywać gotowe procesy dostępnym procesorom.
- **Synchronizacja.** Przy wielu czynnych procesach mających potencjalny dostęp do wspólnych przestrzeni adresowych lub wspólnych zasobów wejścia-wyjścia, trzeba się zatroszczyć o zapewnienie skutecznej synchronizacji. Synchronizacja polega na wymuszaniu wzajemnych wyłączeń i porządkowania zdarzeń.
- **Zarządzanie pamięcią.** Zarządzanie pamięcią w wieloprocesorach musi się uporać ze wszystkimi problemami występującymi w komputerach jednoprocesorowych, które zostały przeanalizowane w rozdz. 8. Ponadto system operacyjny musi wykorzystywać dostępny paralelizm sprzętowy, taki jak pamięci wieloportowe, aby osiągnąć możliwie najwyższą wydajność. Mechanizm stronicowania w wielu procesorach musi być skoordynowany, aby wymusić spójność, gdy kilka procesorów wspólnie użytkuje stronicę lub segment i decyduje się wymienić stronę.
- **Niezawodność i tolerowanie błędów.** System operacyjny powinien zapewnić kontrolowaną degradację w obliczu uszkodzenia procesora. Program szeregujący i inne części systemu operacyjnego muszą rozpoznawać utratę procesora i odpowiednio przebudowywać tabele zarządzania.

Duże komputery SMP

W większości komputerów SMP mających postać komputerów osobistych i stacji roboczych jest stosowane łączenie za pomocą magistrali przedstawione na rys. 18.5. Kształcące jest zapoznanie się z rozwiązaniem alternatywnym, które zostało użyte w najnowszej implementacji rodziny dużych komputerów S/390 IBM [MAK97]. Ogólna organizacja SMP S/390 została przedstawiona na rys. 18.7. Rodzina ta sięga od systemów jednoprocesorowych z jednym modułem pamięci głównej, aż do najbardziej zaawansowanych systemów z dziesięcioma procesorami i czterema modułami pamięci. Konfiguracja obejmuje jeden lub dwa dodatkowe procesory pełniące rolę procesorów wejścia-wyjścia. Kluczowe składniki tej konfiguracji są następujące:

- **Jednostka procesorowa (PU).** Jest to mikroprocesor CISC, w którym najczęściej używane rozkazy są implementowane sprzętowo, pozostałe zaś są wykonywane za pomocą oprogramowania układowego. Każda PU zawiera 64-kilobajtową pamięć podręczną L1, która jest zunifikowana (łączy dane i rozkazy). Rozmiar pamięci podręcznej L1 został wybrany tak, aby mieściła się w mikroukładzie procesora i umożliwiała jednocyklowy dostęp.
- **Pamięć podręczna L2.** Każda pamięć podręczna L2 zawiera 384 KB. Są one zorganizowane w postaci klastrów po dwie pamięci, przy czym każdy klaster obsługuje trzy PU i zapewnia dostęp do całej przestrzeni pamięci głównej.
- **Adapter sieciowy przełączania magistrali (BSN).** Adaptery BSN łączą z sobą pamięci podręczne L2 i pamięć główną. Każdy BSN zawiera również pamięć podręczną trzeciego poziomu (L3) o rozmiarze 2 MB.
- **Moduł pamięci.** Każdy moduł zawiera 8 GB pamięci; łączna pojemność wynosi 32 GB.



Rysunek 18.7. Organizacja S/390 IBM

W konfiguracji SMP S/390 występuje kilka interesujących właściwości, które kolejno przeanalizujemy:

- Połączenia przełączane.
- Wspólne pamięci podręczne L2.
- Pamięć podręczna L3.

Połączenia przełączane

Powszechnie spotykanym rozwiązaniem SMP pełniących rolę komputerów osobistych i stacji roboczych jest pojedyncza, wspólnie użytkowana magistrala (rys. 18.5). Przy takiej organizacji pojedyncza magistrala staje się wąskim gardłem ograniczającym skalowalność konfiguracji (możliwość powiększania rozmiarów). Twórcy S/390 użyli dwóch sposobów, aby zmniejszyć dotkliwość tego problemu. Po pierwsze, pamięć główna została podzielona na cztery odrębne moduły, z których każdy jest wyposażony we własny, bardzo szybki sterownik odpowiedzialny za sterowanie dostępem do pamięci. Przeciętne obciążenie ruchem do pamięci głównej zostało obniżone 4-krotnie dzięki czterem niezależnym ścieżkom prowadzącym do czterech niezależnych części pamięci. Po drugie, połączenie między procesorem (w istocie między pamięciami podręcznymi L2) a pojedynczym modułem pamięci nie ma postaci wspólnej magistrali, a raczej łączy dwupunktowych; każde z nich łączy grupę trzech procesorów poprzez pamięć podręczną L2 z BSN. Z kolei BSN realizuje funkcję przełącznika, który może trasować dane wzdłuż pięciu łączy (cztery łącza L2 i jedno modułu pamięciowego). Jeśli chodzi o cztery łącza L2, BSN łączy cztery łącza fizyczne z jedną magistralą danych logicznych. Zatem sygnał pojawiający się na jednym z czterech łączy L2 jest odbijany do trzech pozostałych łączy L2; właściwość ta jest wymagana dla zapewnienia spójności pamięci.

Zauważmy, że chociaż istnieją tutaj cztery odrębne moduły pamięci, każda PU i każda pamięć podręczna L2 ma tylko dwa porty fizyczne, które mogą się łączyć z pamięcią główną. Jest tak, ponieważ każda pamięć podręczna L2 pobiera dane tylko z połowy pamięci głównej. Do obsłużenia całej pamięci głównej jest potrzebna para pamięci podręcznych i każda PU musi się łączyć z obydwiema pamięciami podręcznymi w takiej parze.

Wspólne pamięci podręczne L2

W typowym, dwupoziomowym układzie pamięci podręcznej SMP każdy procesor dysponuje wydzieloną pamięcią podręczną L1 i wydzieloną pamięcią podręczną L2. Ostatnio wzrosło zainteresowanie wspólną pamięcią L2. W swoich wcześniejszych wersjach SMP S/390, znanych jako generacja 3 (G3), firma IBM użyła wydzielonych pamięci podręcznych L2. W wersjach późniejszych (G4 i G5) została użyta wspólna pamięć podręczna L2. Zmiana ta wynikła z dwóch następujących przesłanek:

1. Przechodząc z G3 do G4, firma IBM podwoiła szybkość mikroprocesorów. Gdyby została zachowana organizacja G3, nastąpiłby znaczny wzrost ruchu na magistrali. Jednocześnie dążono do maksymalnego wykorzystania największej liczby składników G3. Bez znacznego udoskonalenia magistrali adaptery BSN stałyby się wąskim gardłem.
2. Analiza typowych obciążeń roboczych S/390 ujawniła wysoki stopień wspólnego użytkowania rozkazów i danych przez procesory.

Te przesłanki doprowadziły zespół projektujący S/390 G4 do rozważenia użycia jednej lub dwóch pamięci podręcznych L2, z których każda byłaby użytkowana wspólnie przez wiele procesorów (każdy procesor miałby własną pamięć podręczną

L1 wbudowaną w mikroukład). Na pierwszy rzut oka wspólna pamięć podręczna L2 może się wydawać pomysłem chybionym. Dostęp procesorów do pamięci powinien być wolniejszy, ponieważ procesor musi teraz rywalizować o dostęp do pojedynczej pamięci podręcznej L2. Jeśli jednak odpowiednia ilość danych jest w rzeczywistości użytkowana wspólnie przez wiele procesorów, to wspólna pamięć podręczna powinna raczej zwiększyć przepustowość, niż ją ograniczyć. Dane, które są wspólne i znajdują się we wspólnej pamięci podręcznej, mogą być uzyskiwane szybciej, niż gdyby musiała w tym pośredniczyć magistrala.

Jednym z rozwiązań rozważanych przez zespół projektantów S/390 G4 była pojedyncza, duża, w pełni współużytkowana pamięć podręczna, używana przez wszystkie procesory. Chociaż mogłoby to prowadzić do zwiększenia wydajności systemu dzięki wyższej efektywności pamięci podręcznej, takie rozwiązanie wymagałoby całkowitego przeprojektowania istniejącej organizacji magistrali systemowej. Jednak analizy wydajności wykazały, że wprowadzenie wspólnego użytkowania pamięci podręcznej na każdej spośród istniejących magistrali BSN pozwoliłoby utrzymać znaczną część korzyści związanych ze wspólnymi pamięciami podręcznymi przy jednoczesnym ograniczeniu ruchu na magistrali. Wartość wspólnych pamięci podręcznych została potwierdzona przez pomiary wydajności, które wykazały, że wspólna pamięć podręczna poprawiła współczynniki trafienia w porównaniu z pamięciami wydzielonymi zastosowanymi w organizacji G3 [MAK97]. Badania wspólnych pamięci podręcznych w mikroprocesorach SMP o mniejszej skali potwierdziły wartość takiego rozwiązania (np. [NAYF96]).

Pamięć podręczna L3

Kolejną interesującą właściwością SMP S/390 jest użycie trzeciego poziomu pamięci podręcznej (L3)[1]. Pamięci podręczne L3 znajdują się w adapterach BSN, więc każda taka pamięć stanowi bufor między pamięciami podręcznymi L2 a jednym modułem pamięci. Pamięć podręczna L3 redukuje opóźnienie danych, które nie znajdują się w pamięciach podręcznych L1 i L2, a są wymagane przez procesor. Pamięć ta dostarcza dane szybciej, niż gdyby wymagałoby to dostępu do pamięci głównej, a wymagany wiersz pamięci podręcznej byłby już wspólny dla pozostałych procesorów, jednak nie byłby ostatnio używany przez procesor zgłaszający zapotrzebowanie.

W tabeli 18.1 pokazano wyniki badań wydajności trójpoziomowego systemu pamięci podręcznych w przypadku typowego, komercyjnego obciążenia roboczego S/390 z silnie obciążoną pamięcią i magistralą [DOET97][2]. Minusem jest opóźnienie między zgłoszeniem zapotrzebowania na dane do hierarchii pamięciowej a pierwszym uzyskanym 16-bajtowym blokiem danych. Współczynnik trafień w pamięci

[1] W literaturze IBM określa się tę pamięć jako pamięć podręczną L2.5. Używanie takiego określenia nie wydaje się korzystne, ponieważ w istocie stanowi ona trzeci poziom pamięci podręcznej.

[2] Dane dotyczą systemu G3 z wydzielonymi pamięciami podręcznymi L2. Jednak wyniki te są sugestywne, jeśli chodzi o wydajność spodziewaną w przypadku wspólnych pamięci podręcznych L2, takich jak stosowane w S/390 G4 i G5.

podręcznej L1 wynosi 89%, więc pozostałe 11% odniesień do pamięci musi być rozwiązywane na poziomie L2, L3 lub pamięci głównej. Z tych 11% 5% jest rozwiązywane na poziomie L2 itd. Przy trzech poziomach pamięci podręcznej jedynie 3% wymaga dostępu do pamięci głównej. Bez trzeciego poziomu częstość odwołań do pamięci głównej byłaby dwukrotnie większa.

Tabela 18.1. Typowy współczynnik trafień w pamięci podręcznej w SMP S/390

Podsystem pamięci	Opóźnienie dostępu (cykle PU)	Rozmiar pamięci podręcznej	Współczynnik trafień (%)
Pamięć podręczna L1	1	32 KB	89
Pamięć podręczna L2	5	256 KB	5
Pamięć podręczna L3	14	2 MB	3
Pamięć główna	32	8 GB	3

18.3. Spójność pamięci podręcznej i protokół MESI

We współczesnych systemach wieloprocesorowych z każdym procesorem jest zwykle związana jedno- lub dwupoziomowa pamięć podręczna. Organizacja taka ma zasadnicze znaczenie dla uzyskania rozsądnej wydajności. Powoduje ona jednak powstanie problemu znanego jako **problem spójności pamięci podręcznych** (*cache coherence*). Istota tego problemu jest następująca: wiele kopii tych samych danych może jednocześnie występować w różnych pamięciach podręcznych i jeśli procesory mogą swobodnie aktualizować własne kopie, to w rezultacie może powstać niespójny obraz pamięci. W rozdziale 4 zdefiniowaliśmy dwie powszechnie stosowane strategie zapisu:

- **Zapis opóźniony.** Operacje zapisu są zwykle dokonywane wyłącznie w pamięci podręcznej. Pamięć główna jest aktualizowana jedynie wówczas, gdy odpowiedni wiersz pamięci podręcznej zostaje z niej wyrzucony.
- **Zapis jednoczesny.** Wszystkie operacje zapisu są dokonywane zarówno w pamięci głównej, jak i podręcznej, dzięki czemu pamięć główna zawsze zawiera ważne dane.

Jest jasne, że strategia zapisu opóźnionego może prowadzić do niespójności. Jeśli dwie pamięci podręczne zawierają ten sam wiersz, i wiersz ten zostanie zaktualizowany w jednej pamięci podręcznej, druga pamięć podręczna będzie zawierała wartości nieważne. Następujące po tym odczyty tego nieważnego wiersza będą prowadziły do błędnych wyników. Nawet przy zastosowaniu strategii zapisu jednoczesnego może wystąpić niespójność, chyba że pozostałe pamięci podręczne będą monitorowały ruch w pamięci lub otrzymają jakieś bezpośrednie powiadomienie o aktualizacji.

W tym podrozdziale dokonamy krótkiego przeglądu różnych rozwiązań problemu spójności pamięci podręcznych, a następnie skupimy się na najszerzej stosowanym rozwiązaniu: na protokole MESI. Pewna wersja tego protokołu jest używana zarówno w procesorze Pentium 4, jak i w PowerPC.

W przypadku dowolnego protokołu spójności pamięci podręcznej celem jest to, aby ostatnio używane zmienne lokalne trafiły do odpowiedniej pamięci podręcznej i pozostały tam podczas licznych operacji odczytu i zapisu, przy czym używany protokół zapewniłby spójność wspólnie używanych zmiennych, które mogłyby się znaleźć jednocześnie w wielu pamięciach podręcznych. Rozwiązania problemu spójności pamięci podręcznych są ogólnie dzielone na programowe i sprzętowe. W niektórych implementacjach przyjęto strategię obejmującą elementy zarówno programowe, jak i sprzętowe. Mimo to klasyfikacja na rozwiązania programowe i sprzętowe jest nadal pouczająca i powszechnie używana przy rozpatrywaniu strategii zapewniania spójności.

Rozwiązania programowe

W programowych rozwiązaniach spójności pamięci podręcznych próbuje się uniknąć dodatkowych elementów sprzętowych, polegając na kompilatorze i systemie operacyjnym. Rozwiązania programowe są atrakcyjne, ponieważ ciężar wykrywania potencjalnych problemów jest przenoszony z okresu użytkowania na okres kompilowania, a złożoność projektowania jest przenoszona ze sprzętu na oprogramowanie. Jednak rozwiązania programowe z okresu kompilowania na ogół polegają na decyzjach konserwatywnych, co prowadzi do nieefektywnego wykorzystywania pamięci podręcznych.

Mechanizmy spójności oparte na kompilatorze polegają na analizowaniu programu w celu stwierdzenia, które dane mogą stanowić niebezpieczeństwo przy wprowadzaniu ich do pamięci podręcznych, oraz na odpowiednim znakowaniu tych danych. Następnie system operacyjny lub sprzęt zapobiegają kierowaniu tych danych do pamięci podręcznych.

Najprostszym rozwiązaniem jest zapobieganie temu, żeby jakiekolwiek wspólne zmienne były kierowane do pamięci podręcznych. Jest to jednak rozwiązanie zbyt konserwatywne, ponieważ wspólne struktury danych mogą być w pewnych okresach używane na zasadzie wyłączności, natomiast w pozostałych okresach mogą być efektywnie odczytywane. Spójność pamięci podręcznych stanowi problem tylko wówczas, gdy przynajmniej jeden proces może aktualizować zmienną i przynajmniej jeden inny proces może sięgać po tę zmienną.

W bardziej efektywnych rozwiązaniach program jest analizowany w celu określenia okresów bezpiecznych dla wspólnych zmiennych. Następnie kompilator umieszcza w generowanym programie rozkaz, który wymusza spójność pamięci podręcznych podczas okresów krytycznych. Opracowano wiele metod przeprowadzania takiej analizy i wymuszania spójności; opisano je w [LILJ93] i w [STEN90].

Rozwiązania sprzętowe

Rozwiązania sprzętowe są na ogół określane jako protokoły spójności pamięci podręcznych. Polegają one na dynamicznym rozpoznawaniu warunków potencjalnej niespójności w czasie użytkowania systemu. Ponieważ problem jest rozwiązywany

tylko wtedy, kiedy rzeczywiście powstaje, wykorzystanie pamięci podręcznych jest bardziej efektywne, co prowadzi do poprawy wydajności w stosunku do rozwiązań programowych. Ponadto rozwiązania te są widzialne zarówno dla programisty, jak i dla kompilatora, co redukuje ciężar opracowywania oprogramowania.

Rozwiązania sprzętowe różnią się wieloma szczegółami, między innymi tym, gdzie jest przechowywana informacja o stanie bloków danych, jak ta informacja jest zorganizowana, gdzie jest wymuszana spójność i jakie są mechanizmy wymuszania. Rozwiązania sprzętowe mogą być podzielone na dwie ogólne kategorie: protokoły katalogowe i protokoły podglądania.

Protokoły katalogowe

Protokoły katalogowe polegają na zbieraniu i przechowywaniu informacji o tym, gdzie rezydują kopie bloków danych. Zwykle istnieje centralny sterownik będący częścią sterownika pamięci głównej, a katalog jest przechowywany w pamięci głównej. Katalog zawiera globalną informację o stanie zawartości różnych lokalnych pamięci podręcznych. Gdy sterownik określonej pamięci podręcznej zgłasza zapotrzebowanie, sterownik centralny sprawdza i wydaje niezbędne rozkazy zapewniające przesyłanie danych między pamięcią główną a pamięciami podręcznymi lub między pamięciami podręcznymi. Jest on także odpowiedzialny za utrzymywanie i ciągłe aktualizowanie informacji o stanie; informacja o każdym działaniu lokalnym, które może wpłynąć na globalny stan bloku danych, musi trafić do sterownika centralnego.

Sterownik zachowuje zwykle informację o tym, które procesory mają kopię których bloków. Zanim procesor będzie mógł dokonać operacji zapisu w lokalnej kopii bloku, musi zgłosić do sterownika centralnego zapotrzebowanie na wyłączny dostęp do tego bloku. Przed udzieleniem zgody na wyłączny dostęp, sterownik wysyła wiadomość do wszystkich procesorów przechowujących w pamięci podręcznej kopię tego bloku, zmuszając każdy procesor do unieważnienia kopii. Po otrzymaniu potwierdzeń unieważnienia od każdego procesora, sterownik udziela prawa wyłącznego dostępu procesorowi zgłaszającemu zapotrzebowanie. Gdy inny procesor próbuje odczytać blok udostępniony na zasadzie wyłączności innemu procesorowi, wysyła zawiadomienie o chybieniu do sterownika. Sterownik wydaje wtedy procesorowi dysponującemu blokiem rozkaz zapisu do pamięci głównej. Teraz blok może być jednocześnie odczytywany przez oba procesory.

Wadą rozwiązań katalogowych jest wąskie gardło, jakie może wystąpić w centralnym sterowaniu, oraz obciążenie komunikacją między różnymi sterownikami pamięci podręcznych a sterownikiem centralnym. Są one jednak efektywne w dużych systemach o wielu magistralach lub z innym złożonym układem połączeń.

Protokoły podglądania

W protokołach podglądania odpowiedzialność za utrzymywanie spójności pamięci podręcznych jest rozkładana na wszystkie sterowniki tych pamięci w wieloprocesorze. Pamięć podręczna musi rozpoznać sytuację, w której przechowywany przez nią

blok występuje również w innych pamięciach podręcznych. Gdy jest aktualizowany blok wspólny, muszą być o tym powiadomione pozostałe pamięci podręczne za pośrednictwem pewnego mechanizmu rozgłaszania. Każdy sterownik pamięci podręcznej może przeglądać („podgląda") sieć w poszukiwaniu takich powiadomień i odpowiednio reagować.

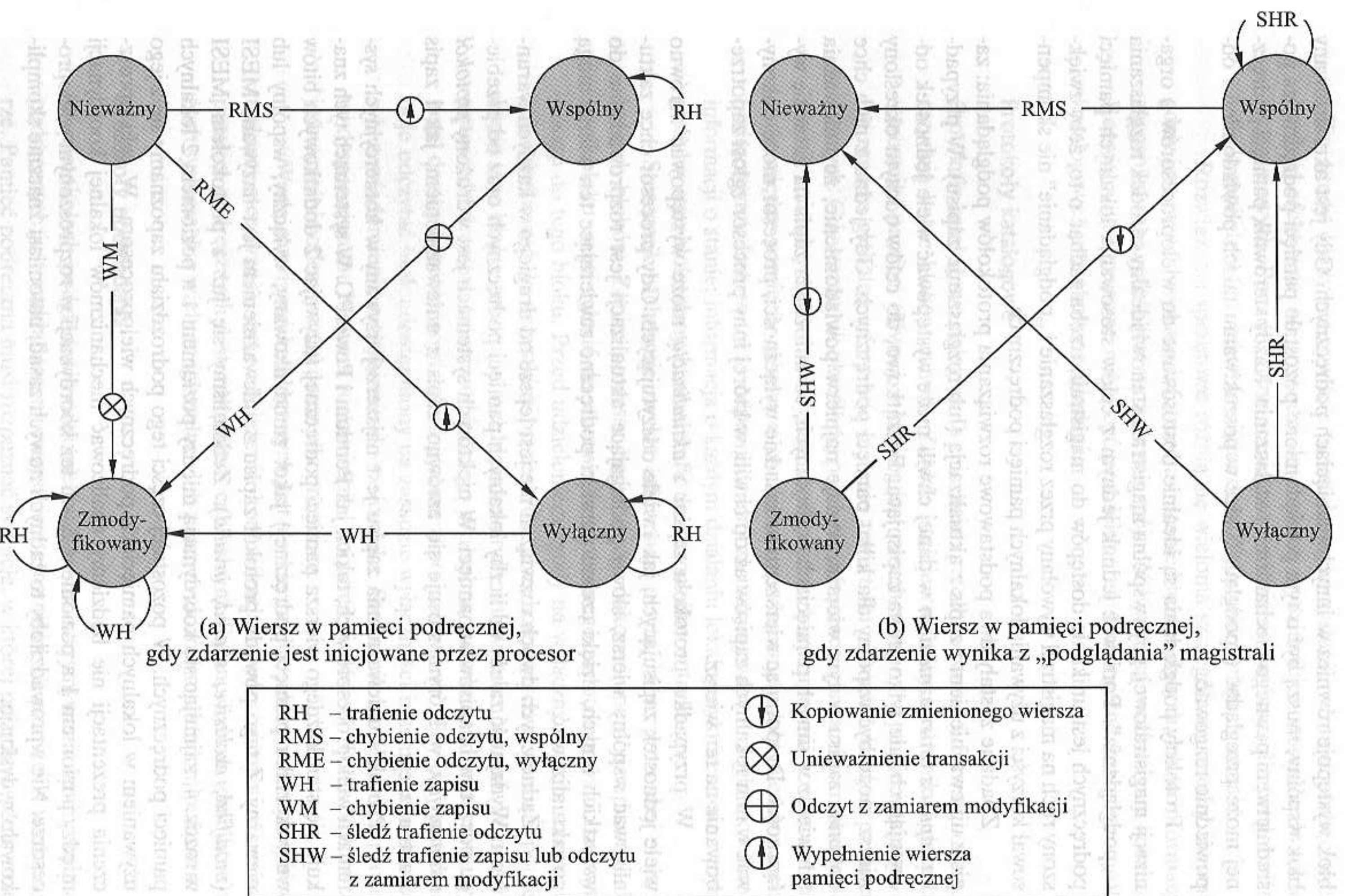
Protokoły podglądania są idealnie dostosowane do wieloprocesorów o organizacji magistralowej, gdyż wspólna magistrala stanowi idealny ośrodek rozgłaszania i „podglądania". Ponieważ jednak jednym z celów stosowania lokalnych pamięci podręcznych jest unikanie dostępów do magistrali, trzeba zadbać o to, żeby zwiększony ruch na magistrali wywołany przez rozgłaszanie i „podglądanie" nie skompensował korzyści z używania lokalnych pamięci podręcznych.

Zbadane zostały dwa podstawowe rozwiązania protokołów podglądania: zapis z unieważnieniem i zapis z aktualizacją (lub rozgłaszanie zapisu). W przypadku *zapisu z unieważnieniem* w danej chwili może występować wiele jednostek odczytujących, ale tylko jedna zapisująca. Początkowo, do celów odczytu określony wiersz może być wspólny dla kilku pamięci podręcznych. Gdy jedna z nich chce dokonać zapisu w tym wierszu, wydaje najpierw powiadomienie, że unieważnia ten wiersz w innych pamięciach podręcznych, co umożliwia zapis na zasadzie wyłączności. Dysponując wierszem na zasadzie wyłączności, procesor może dokonywać w nim lokalnych zapisów aż do chwili, w której inny procesor zgłosi zapotrzebowanie na ten wiersz.

W przypadku protokołu *zapisu z aktualizacją* może występować zarówno wiele jednostek zapisujących, jak i wiele odczytujących. Gdy procesor chce zaktualizować wspólny wiersz, słowo podlegające aktualizacji jest rozprowadzane do wszystkich innych, dzięki czemu pamięci podręczne zawierające ten wiersz mogą go zaktualizować.

Żadne z tych dwóch rozwiązań nie jest lepsze od drugiego w każdych warunkach. Wydajność zależy od liczby lokalnych pamięci podręcznych oraz od przebiegu odczytów i zapisów w pamięci. W niektórych systemach jest wdrożony *protokół adaptacyjny*, w którym stosuje się zarówno zapis z unieważnieniem, jak i zapis z aktualizacją.

Protokół unieważniania zapisu jest najszerzej używany w komercyjnych systemach wieloprocesorowych, takich jak Pentium i PowerPC. W systemach tych znakuje się stan każdego wiersza pamięci podręcznej (używając 2 dodatkowych bitów we wskaźniku pamięci podręcznej) jako: zmodyfikowany, wyłączny, wspólny lub nieważny. Z tego powodu protokół zapisu z unieważnieniem jest nazywany MESI (*modified*, *exclusive*, *shared*, *invalid*). Zetknęliśmy się już z protokołem MESI w rozdz. 4, zajmując się koordynacją między poziomem 1 a poziomem 2 lokalnych pamięci podręcznych. W pozostałej części tego podrozdziału zapoznamy się z jego używaniem w lokalnych pamięciach podręcznych wieloprocesora. W celu uproszczenia prezentacji nie będziemy analizować mechanizmów lokalnej koordynacji między poziomem 1 a poziomem 2, ani też koordynacji w rozproszonym wieloprocesorze. Nie wprowadziłoby to żadnych nowych zasad, natomiast znacznie skomplikowałoby dyskusję.



Rysunek 18.8. Graf przejść MESI

Protokół MESI

W celu zapewnienia spójności pamięci podręcznej w SMP w pamięciach podręcznych danych jest zwykle stosowany protokół znany jako MESI. W przypadku protokołu MESI pamięć podręczna danych zawiera w swoim wskaźniku 2 bity stanu; każdy wiersz może znajdować się w jednym z czterech stanów:

- **Zmodyfikowany.** Wiersz pamięci podręcznej został zmodyfikowany (różni się od odpowiednika w pamięci głównej) i jest osiągalny tylko w tej pamięci podręcznej.
- **Wyłączny.** Wiersz pamięci podręcznej jest taki sam jak w pamięci głównej i nie występuje w innych pamięciach podręcznych.
- **Wspólny.** Wiersz w pamięci podręcznej jest taki sam jak w pamięci głównej i może występować w innej pamięci podręcznej.
- **Nieważny.** Wiersz pamięci podręcznej nie zawiera danych ważnych.

Tabela 18.2. Stany wiersza pamięci podręcznej MESI

	M Zmodyfikowany	E Wyłączny	S Wspólny	I Nieważny
Czy ten wiersz pamięci podręcznej jest ważny?	Tak	Tak	Tak	Nie
Kopia pamięci jest...	nieaktualna	ważna	ważna	–
Czy istnieją kopie w innych pamięciach podręcznych?	Nie	Nie	Być może	Być może
Zapis w tym wierszu...	Nie trafia do magistrali	Nie trafia do magistrali	Trafia do magistrali i aktualizuje pamięć podręczną	Trafia bezpośrednio do magistrali

W tabeli 18.2 przedstawiono cechy poszczególnych czterech stanów. Na rysunku 18.8 został pokazany wykres stanów dla protokołu MESI. Pamiętajmy, że każdy wiersz pamięci podręcznej ma własne bity stanu i tym samym własną „świadomość" stanu. Na rysunku 18.8a pokazano przejścia, które następują w wyniku zdarzeń podejrzanych na wspólnej magistrali. Ta prezentacja odrębnych wykresów stanu dla działań inicjowanych przez procesor i inicjowanych przez magistralę pomaga wyjaśnić logikę protokołu MESI. W każdej chwili wiersz pamięci podręcznej znajduje się w jednym stanie. Jeśli następne zdarzenie pochodzi z dołączonego procesora, to przejście zachodzi zgodnie z rys. 18.8a, a jeśli następne zdarzenie pochodzi z magistrali, przejście zachodzi zgodnie z rys. 18.8b. Zapoznajmy się z tymi przejściami bardziej szczegółowo.

Chybienie odczytu

Gdy w lokalnej pamięci podręcznej następuje chybienie odczytu, procesor inicjuje odczyt w pamięci w celu odczytania wiersza w pamięci głównej zawierającego brakujący adres. Procesor umieszcza sygnał na magistrali alarmujący wszystkie pozo-

stałe jednostki procesor/pamięć podręczną, żeby śledziły transakcję. Wynikają stąd następujące możliwe sytuacje:

- Jeśli jedna z pozostałych pamięci podręcznych ma czystą (niezmodyfikowaną od czasu wczytania z pamięci) kopię wiersza w stanie wyłącznym, to sygnalizuje w odpowiedzi, że dysponuje tym blokiem. Odpowiadający procesor zmienia następnie stan swojej kopii z wyłącznego na wspólny, a procesor inicjujący wczytuje wiersz z pamięci głównej i zmienia stan wiersza w swojej pamięci podręcznej z nieważnego na wspólny.
- Jeśli jedna lub wiele pamięci podręcznych ma czystą kopię wiersza w stanie wspólnym, to każda z nich sygnalizuje ten stan. Procesor inicjujący wczytuje blok i zmienia stan wiersza w swojej pamięci podręcznej z nieważnego na wspólny.
- Jeśli jedna z pozostałych pamięci podręcznych ma zmodyfikowaną kopię wiersza, to ta pamięć podręczna blokuje odczyt w pamięci głównej i przekazuje wiersz poprzez wspólną magistralę. Następnie odpowiadająca pamięć podręczna zmienia stan swoich wierszy ze zmodyfikowanego na wspólny[3].
- Jeśli żadna inna pamięć podręczna nie ma kopii wiersza (czystej ani zmodyfikowanej), to w odpowiedzi nie są przekazywane żadne sygnały. Procesor inicjujący wczytuje wiersz i zmienia stan wiersza w swojej pamięci podręcznej z nieważnego na wyłączny.

Trafienie odczytu

Gdy następuje trafienie odczytu wiersza w lokalnej pamięci podręcznej, procesor po prostu odczytuje wymagane dane. Nie następuje zmiana stanu: stan pozostaje zmodyfikowany, wspólny lub wyłączny.

Chybienie zapisu

Gdy następuje chybienie zapisu w lokalnej pamięci podręcznej, procesor inicjuje odczyt pamięci w celu wczytania z pamięci głównej wiersza zawierającego brakujący adres. Aby to osiągnąć, procesor podaje na magistralę sygnał oznaczający odczyt z zamiarem modyfikacji (*read-with-intent-to-modify* – RWITM). Wiersz po załadowaniu jest natychmiast znakowany jako zmodyfikowany. W odniesieniu do pozostałych pamięci podręcznych ładowanie wiersza danych poprzedzają dwa możliwe scenariusze.

Po pierwsze, jedna z pozostałych pamięci podręcznych może mieć zmodyfikowaną kopię tego wiersza (stan = zmodyfikowany). W takim przypadku zaalarmowany procesor sygnalizuje procesorowi inicjującemu, że inny procesor ma zmo-

[3] W niektórych implementacjach pamięć podręczna zawierająca zmodyfikowany wiersz sygnalizuje procesorowi inicjującemu celowość ponownej próby. W tym samym czasie procesor dysponujący zmodyfikowaną kopią przejmuje sterowanie magistralą, zapisuje zmodyfikowany blok w pamięci głównej i zmienia stan tego bloku w swojej pamięci podręcznej ze zmodyfikowanego na wspólny. Następnie procesor inicjujący dokonuje ponownej próby i stwierdza, że jeden lub wiele procesorów ma czystą kopię bloku w stanie wspólnym, o czym była mowa w poprzednim punkcie.

dyfikowaną kopię wiersza. Procesor inicjujący oddaje sterowanie magistralą i czeka. Drugi procesor przejmuje sterowanie magistralą, zapisuje zmodyfikowany wiersz w pamięci głównej i zmienia stan wiersza w swojej pamięci podręcznej na nieważny (ponieważ procesor inicjujący zamierza zmodyfikować ten wiersz). Następnie procesor inicjujący ponownie przekazuje na magistralę sygnał RWITM, po czym zapisuje wiersz w pamięci głównej.

Drugi scenariusz występuje wtedy, gdy żadna inna pamięć podręczna nie ma zmodyfikowanej kopii wymaganego wiersza. W tym przypadku w odpowiedzi nie nadchodzi żaden sygnał i procesor inicjujący przystępuje do wczytania i modyfikowania wiersza. W tym samym czasie, jeśli jeden lub wiele procesorów ma czystą kopię wiersza w stanie wspólnym, to każdy procesor unieważnia swoją kopię wiersza. Jeśli jeden z procesorów ma czystą kopię wiersza w stanie wyłącznym, to unieważnia ją.

Trafienie zapisu

Gdy następuje trafienie zapisu do wiersza znajdującego się w lokalnej pamięci podręcznej, wynik zależy od bieżącego stanu tego wiersza w pamięci podręcznej:

- **Wspólny.** Przed przeprowadzeniem aktualizacji procesor musi uzyskać prawo wyłączności dostępu do bloku. Procesor sygnalizuje swoje intencje za pośrednictwem magistrali. Każdy procesor, który ma wspólną kopię bloku w swojej pamięci podręcznej, zmienia stan bloku ze wspólnego na nieważny. Następnie procesor inicjujący dokonuje aktualizacji i zmienia stan kopii bloku ze wspólnego na zmodyfikowany.
- **Wyłączny.** Procesor ma już wyłączną kontrolę nad blokiem, więc po prostu dokonuje aktualizacji i zmienia stan kopii bloku z wyłącznego na zmodyfikowany.
- **Zmodyfikowany.** Procesor ma już wyłączną kontrolę nad blokiem i blok ten jest zaznaczony jako zmodyfikowany, więc po prostu dokonuje aktualizacji.

Spójność pamięci podręcznych L1-L2

Dotychczas opisywaliśmy protokoły spójności pamięci podręcznych w kategoriach współdziałania pamięci dołączonych do tej samej magistrali lub do innego układu połączeń SMP. Tymi pamięciami podręcznymi są zwykle pamięci L2, natomiast każdy procesor ma również pamięć podręczną L1, która nie łączy się bezpośrednio z magistralą i dlatego nie jest w stanie angażować się w protokół podglądania. Potrzebny jest więc sposób zachowywania integralności danych na obydwu poziomach pamięci podręcznych oraz we wszystkich pamięciach podręcznych występujących w konfiguracji SMP.

Sposób ten polega na rozszerzeniu protokołu MESI (lub jakiegokolwiek innego protokołu spójności pamięci podręcznych) na pamięci L1. W wyniku takiego poszerzenia każdy wiersz pamięci podręcznej L1 zawiera bity wskazujące stan. Cel jest w istocie następujący: w odniesieniu do każdego wiersza, który jest obecny zarówno w pamięci L2, jak i w odpowiadającej jej pamięci podręcznej L1, stan wiersza

L1 powinien naśladować stan wiersza L2. Prostym sposobem osiągnięcia tego jest zastosowanie strategii jednoczesnego zapisu w pamięci L1; chodzi tu o jednoczesny zapis w pamięci L2, nie zaś w pamięci głównej. Zapis jednoczesny w L1 wymusza przekazywanie wszelkich modyfikacji wierszy w L1 do pamięci podręcznej L2, i tym samym uczynienie ich widzialnymi dla pozostałych pamięci L2. Zastosowanie metody zapisu jednoczesnego w L1 sprawia, że zawartość L1 musi być podzbiorem zawartości L2. To z kolei sugeruje, że pamięć L2 powinna być co najmniej w tym samym stopniu skojarzeniowa, co pamięć L1. Metoda zapisu jednoczesnego w L1 została przyjęta w SMP S/390 IBM.

Jeśli w pamięci podręcznej L1 zostałaby użyta metoda zapisu opóźnionego, zależności miedzy obiema pamięciami podręcznymi stałyby się znacznie bardziej skomplikowane. Istnieje kilka rozwiązań zachowania spójności. Rozwiązanie użyte w Pentium II zostało szczegółowo opisane w [SHAN98].

18.4. Klastry

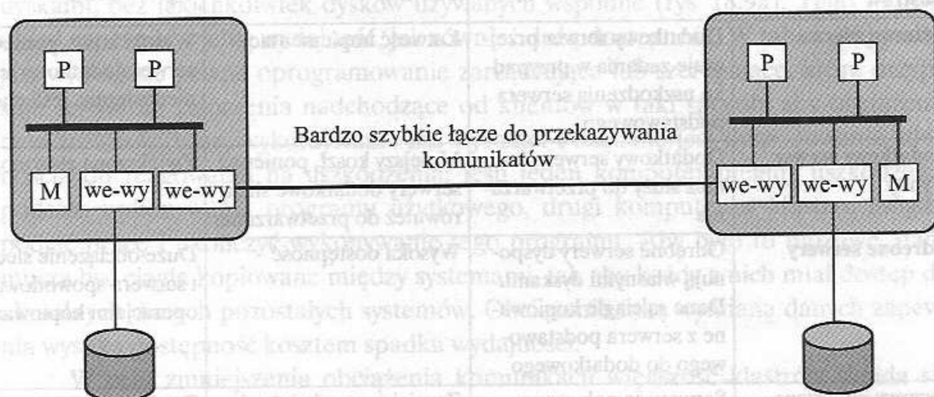
Jedna z najnowszych dziedzin projektowania systemów komputerowych to tworzenie klastrów. Stanowi ono rozwiązanie alternatywne w stosunku do wieloprzetwarzania symetrycznego; zapewnia większą wydajność i lepszą dostępność, dlatego jest szczególnie atrakcyjne w serwerach. Klaster (*cluster*) możemy zdefiniować jako grupę połączonych ze sobą, kompletnych komputerów tworzących razem jednolity zasób obliczeniowy, który może sprawiać wrażenie, że jest jedną maszyną. Określenie *kompletny komputer* oznacza system zdolny do pracy samodzielnej, poza klastrem; w literaturze każdy komputer w klastrze jest zwykle określany jako *węzeł*.

[BREW97] wymienia cztery korzyści, jakie mogą być osiągnięte dzięki tworzeniu klastrów. Można je traktować jako cele lub wymagania projektowe:

- **Skalowalność bezwzględna.** Jest możliwe tworzenie wielkich klastrów, które daleko przekraczają możliwości nawet największych maszyn samodzielnych. Klaster może zawierać dziesiątki komputerów, z których każdy jest wieloprocesorem.
- **Skalowalność przyrostowa.** Klastry są konfigurowane tak, że jest możliwe dodawanie do nich nowych systemów w postaci niewielkich przyrostów. Użytkownik może zatem rozpocząć od sytemu o umiarkowanych rozmiarach i poszerzać go stosownie do potrzeb, bez konieczności dokonywania poważnej modernizacji polegającej na zastępowaniu niewielkiego systemu większym.
- **Wysoka dostępność.** Ponieważ każdy węzeł klastra jest samodzielnym komputerem, uszkodzenie jednego węzła nie oznacza utraty możliwości obsługi. W wielu produktach odpowiednie zmiany są dokonywane automatycznie dzięki oprogramowaniu.
- **Korzystny stosunek ceny do wydajności.** Dzięki stosowaniu typowych składników dostępnych w handlu, jest możliwe złożenie klastra o zdolności obliczeniowej co najmniej takiej samej, jak pojedynczego, dużego komputera, przy znacznie niższym koszcie.

Konfiguracja klastra

W literaturze klastry są klasyfikowane na różne sposoby. Być może najprostsza klasyfikacja jest oparta na tym, czy komputery w klastrze dzielą się dostępem do tych samych dysków. Na rysunku 18.9a został pokazany dwuwęzłowy klaster, w którym jedynym połączeniem wzajemnym jest łącze o dużej szybkości, które może być użyte do wymiany komunikatów w celu koordynowania pracy klastra. Łącze to może być siecią lokalną (LAN), która jest wspólnie użytkowana z innymi komputerami nie należącymi do danego klastra, lub może być specjalistyczną strukturą połączeń. W tym ostatnim przypadku jeden lub wiele komputerów w klastrze może dysponować łączem z siecią lokalną lub rozległą (WAN), dzięki czemu istnieje połączenie między klastrem-serwerem a zdalnymi systemami-klientami. Zauważmy, że na tym rysunku każdy komputer został przedstawiony jako wieloprocesorowy. Nie jest to konieczne, jednak umożliwia zwiększenie zarówno wydajności, jak i dostępności.



(a) Serwer rezerwowy bez współużytkowanych dysków



(b) Współużytkowany dysk

Rysunek 18.9. Konfiguracja klastra

W prostej klasyfikacji zilustrowanej na rys. 18.9 drugim rozwiązaniem jest klaster o współużytkowanym dysku. W tym przypadku na ogół nadal istnieje łącze służące do przekazywania komunikatów między węzłami. Ponadto istnieje podsystem dyskowy, który jest bezpośrednio połączony z należącymi do klastra komputerami. Na rysunku tym wspólnym podsystemem jest system RAID. Zastosowanie rozwiązania RAID lub podobnej technologii redundancji dysków jest w klastrach powszechne, przez co wysoka dostępność osiągana dzięki obecności wielu komputerów nie jest narażana na szwank przez wspólny dysk, którego uszkodzenie pociągnęłoby za sobą degradację całego systemu.

Jaśniejszy obraz zasięgu możliwości klastrów można uzyskać, zapoznając się z różnymi rozwiązaniami funkcjonalnymi. W tabeli 18.3 została przedstawiona klasyfikacja funkcjonalna klastrów, którą się teraz zajmiemy.

Tabela 18.3. Metody tworzenia klastrów: korzyści i ograniczenia

Metoda tworzenia klastrów	Opis	Korzyści	Ograniczenia
Rezerwa bierna	Dodatkowy serwer przejmuje zadania w przypadku uszkodzenia serwera podstawowego	Łatwość implementacji	Wysoki koszt, ponieważ serwer dodatkowy jest niedostępny, jeśli chodzi o przetwarzanie
Dodatkowy serwer czynny	Dodatkowy serwer również służy do przetwarzania	Mniejszy koszt, ponieważ serwery dodatkowe służą również do przetwarzania	Zwiększona złożoność
Odrębne serwery	Odrębne serwery dysponują własnymi dyskami. Dane są ciągle kopiowane z serwera podstawowego do dodatkowego	Wysoka dostępność	Duże obciążenie sieci i serwera spowodowane operacjami kopiowania
Serwery połączone z dyskami	Serwery są połączone z tymi samymi dyskami, jednak każdy z nich ma własne dyski. Gdy jeden serwer ulega uszkodzeniu, dyski są przejmowane przez drugi serwer	Zmniejszone obciążenie sieci i serwera w wyniku wyeliminowania operacji kopiowania	Zwykle wymaga tworzenia kopii zwierciadlanych lub technologii RAID w celu skompensowania niebezpieczeństwa uszkodzenia dysku
Serwery współużytkujące dyski	Wiele serwerów jednocześnie ma dostęp do dysków	Niskie obciążenie sieci i serwera. Zmniejszone niebezpieczeństwo przestoju spowodowanego przez uszkodzenie dysku	Wymaga oprogramowania zarządzania blokadami. Zwykle jest stosowane tworzenie kopii zwierciadlanych lub technologia RAID

Powszechnie znana, starsza metoda nazywana **bierną rezerwą**, polega po prostu na tym, że jeden komputer jest obciążony całym przetwarzaniem, podczas gdy drugi pozostaje nieczynny, gotów do przejęcia pracy w przypadku uszkodzenia pierwszego. W celu skoordynowania tych komputerów, system czynny (podstawo-

wy) okresowo wysyła komunikat o swoim działaniu („bicie serca") do maszyny rezerwowej. Gdy te komunikaty przestają napływać, komputer rezerwowy zakłada, że serwer podstawowy uległ uszkodzeniu i przejmuje jego zadania. Takie rozwiązanie zwiększa dostępność, lecz nie poprawia wydajności. Co więcej, jeśli informacje wymieniane między dwoma systemami dotyczą jedynie informacji o funkcjonowaniu, to komputer rezerwowy zapewnia rezerwę funkcjonalną, jednak nie ma dostępu do baz danych, którymi zarządzał komputer podstawowy.

Rezerwa bierna nie jest zwykle zaliczana do klastrów. Wyrażenie *klaster* jest zarezerwowane dla wielu wzajemnie połączonych komputerów, z których wszystkie zajmują się przetwarzaniem, zachowując wobec świata zewnętrznego obraz pojedynczego systemu. W odniesieniu do takiej konfiguracji jest zwykle używane określenie **dodatkowy serwer czynny**. Można przyjąć następującą klasyfikację klastrów: odrębne serwery, serwery nie współużytkujące niczego oraz serwery współużytkujące pamięć.

W jednym z rozwiązań każdy komputer jest **odrębnym serwerem** z własnymi dyskami, bez jakichkolwiek dysków używanych wspólnie (rys. 18.9a). Tego rodzaju układ zapewnia wielką wydajność, jak również dużą dostępność. W takim przypadku jest wymagane pewne oprogramowanie zarządzające lub szeregujące, które przypisuje serwerom zgłoszenia nadchodzące od klientów w taki sposób, aby obciążenie było zrównoważone, wykorzystanie zaś wysokie. Pożądane jest dysponowanie zdolnością do reagowania na uszkodzenia; jeśli jeden komputer uległby uszkodzeniu podczas wykonywania programu użytkowego, drugi komputer w klastrze mógłby podjąć pracę i ukończyć wykonywanie tego programu. Aby było to możliwe, dane muszą być ciągle kopiowane między systemami, tak aby każdy z nich miał dostęp do aktualnych danych pozostałych systemów. Obciążenie taką wymianą danych zapewnia wysoką dostępność kosztem spadku wydajności.

W celu zmniejszenia obciążenia komunikacji większość klastrów składa się obecnie z serwerów połączonych ze wspólnymi dyskami (rys. 18.9b). W odmianie tego rozwiązania, określanej jako **brak współużytkowania czegokolwiek**, wspólne dyski są dzielone na części, a każdą z tych części dysponuje pojedynczy komputer. Jeśli taki pojedynczy komputer ulegnie uszkodzeniu, klaster musi być ponownie skonfigurowany w taki sposób, aby jakiś inny komputer przejął dysponowanie częścią dysku kontrolowaną uprzednio przez uszkodzony komputer.

Jest również możliwe dysponowanie wieloma komputerami jednocześnie współużytkującymi te same dyski (rozwiązanie określane jako **współużytkowany dysk**); wówczas każdy komputer ma dostęp do wszystkich części dysku. Wymaga to zastosowania pewnego rodzaju rozwiązania blokującego, które zapewniałoby, że w danej chwili do określonych danych mógłby sięgać tylko jeden komputer.

Problemy projektowania systemu operacyjnego

Pełne wykorzystanie klastrowej konfiguracji sprzętowej wymaga pewnych poszerzeń systemu operacyjnego w stosunku do systemu przeznaczonego dla pojedynczych komputerów.

Zarządzanie uszkodzeniami

To, w jaki sposób klaster postępuje z uszkodzeniami, jest uzależnione od zastosowanej metody tworzenia klastra (tabela 18.3). Na ogół używane są dwa rozwiązania: klastry o wielkiej dostępności i klastry tolerancyjne wobec uszkodzeń. Klaster o wielkiej dostępności oferuje wysokie prawdopodobieństwo, że wszystkie zasoby będą gotowe do użytku. Jeśli wystąpi uszkodzenie, takie jak załamanie systemu lub utrata części dysku, to bieżąco przetwarzane zapytania są tracone. Wszelkie utracone zapytania, jeśli zostaną wznowione, będą obsłużone przez inny komputer w klastrze. Jednak system operacyjny klastra nie gwarantuje zachowania stanu częściowo wykonanych transakcji. Wymagałoby to manipulowania na poziomie aplikacji.

Klaster tolerancyjny wobec uszkodzeń zapewnia, że wszystkie zasoby są zawsze dostępne. Osiąga się to dzięki redundancyjnym dyskom współużytkowanym i mechanizmom tworzenia rezerwowych kopii transakcji nieukończonych oraz powierzania transakcji ukończonych.

Funkcja przełączania aplikacji i zasobów danych z systemu uszkodzonego do innego systemu w klastrze jest określana jako **przejmowanie danych** (*failover*). Funkcją pokrewną jest przywracanie aplikacji i zasobów danych do systemu początkowego, gdy zostanie on naprawiony; jest to określane mianem **przywracania** (*failback*). Przywracanie może być zautomatyzowane, jednak jest to pożądane tylko wówczas, gdy uszkodzenie rzeczywiście zostało naprawione, a jego powtórzenie się jest mało prawdopodobne. Jeśli tak nie jest, automatyczne przywracanie mogłoby spowodować wielokrotne przekazywanie zasobów między komputerami, czego wynikiem byłby spadek wydajności i trudności z powrotem systemu do stanu normalnego.

Równoważenie obciążenia

Klaster wymaga skutecznej zdolności do równoważenia obciążenia między dostępnymi komputerami. Wiąże się z tym wymaganie, aby klaster umożliwiał przyrostową skalowalność. Gdy do klastra zostanie dodany nowy komputer, rozwiązanie służące do równoważenia obciążenia powinno automatycznie uwzględniać nowy komputer podczas szeregowania. Środki programowo-sprzętowe muszą rozpoznawać, że usługi mogą się ukazywać w różnych systemach wchodzących w skład klastra i że mogą migrować z jednego systemu do drugiego.

Równoległe wykonywanie programów użytkowych

W pewnych przypadkach skuteczne wykorzystywanie klastrów wymaga równoległego wykonywania pojedynczych programów aplikacyjnych. [KAPP00] wymienia trzy ogólne rozwiązania tego problemu:

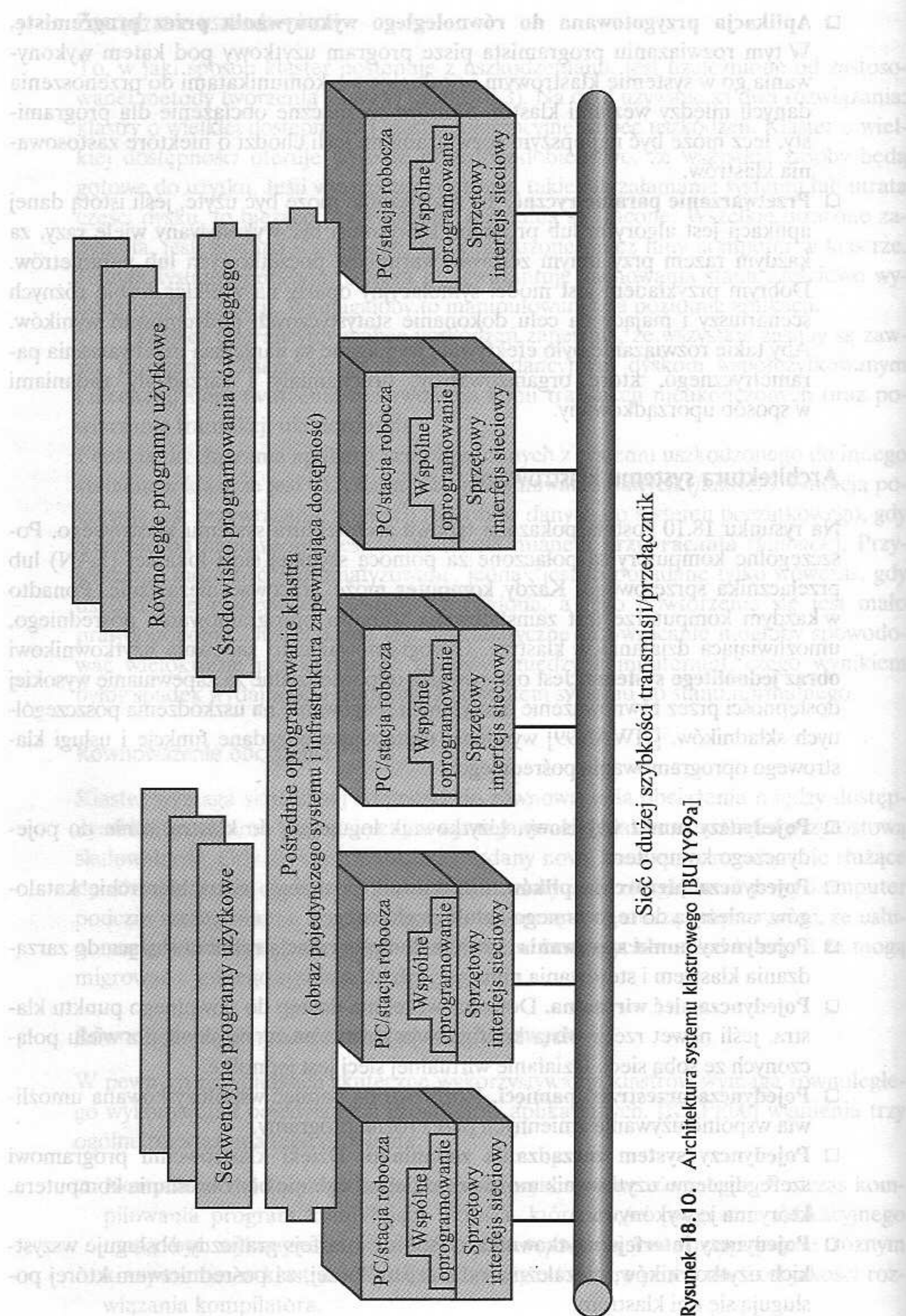
- **Kompilator dzielący aplikacje na części wykonywane równolegle.** Podczas kompilowania program kompilujący określa, które części programu aplikacyjnego mogą być wykonywane równolegle. Następnie są one przypisywane różnym komputerom w klastrze. Wydajność zależy od natury problemu i od jakości rozwiązania kompilatora.

- **Aplikacja przygotowana do równoległego wykonywania przez programistę.** W tym rozwiązaniu programista pisze program użytkowy pod kątem wykonywania go w systemie klastrowym i posługuje się komunikatami do przenoszenia danych między węzłami klastra. Stanowi to znaczne obciążenie dla programisty, lecz może być najlepszym rozwiązaniem, jeśli chodzi o niektóre zastosowania klastrów.
- **Przetwarzanie parametryczne.** To rozwiązanie może być użyte, jeśli istotą danej aplikacji jest algorytm lub program, który musi być wykonywany wiele razy, za każdym razem przy innym zestawie warunków początkowych lub parametrów. Dobrym przykładem jest model symulacyjny oparty na wielkiej liczbie różnych scenariuszy i mający na celu dokonanie statystycznych podsumowań wyników. Aby takie rozwiązanie było efektywne, wymagane są narzędzia przetwarzania parametrycznego, które organizowałyby, uruchamiały i zarządzały zadaniami w sposób uporządkowany.

Architektura systemu klastrowego

Na rysunku 18.10 została pokazana typowa architektura systemu klastrowego. Poszczególne komputery są połączone za pomocą szybkiej sieci lokalnej (LAN) lub przełącznika sprzętowego. Każdy komputer może pracować niezależnie. Ponadto w każdym komputerze jest zainstalowana warstwa oprogramowania pośredniego, umożliwiająca działanie w klastrze. Oprogramowanie to zapewnia użytkownikowi **obraz jednolitego systemu.** Jest ono również odpowiedzialne za zapewnianie wysokiej dostępności przez równoważenie obciążenia i reagowanie na uszkodzenia poszczególnych składników. [HWAN99] wymienia następujące pożądane funkcje i usługi klastrowego oprogramowania pośredniego:

- **Pojedynczy punkt wejściowy.** Użytkownik loguje się do klastra, a nie do pojedynczego komputera.
- **Pojedyncza hierarchia plików.** Użytkownik postrzega jedną hierarchię katalogów, należąca do tego samego katalogu głównego.
- **Pojedynczy punkt sterowania.** Istnieje domyślna stacja robocza służąca do zarządzania klastrem i sterowania nim.
- **Pojedyncza sieć wirtualna.** Dowolny węzeł ma dostęp do dowolnego punktu klastra, jeśli nawet rzeczywista konfiguracja klastra może się składać z wielu połączonych ze sobą sieci. Działanie wirtualnej sieci jest jednolite.
- **Pojedyncza przestrzeń pamięci.** Rozproszona pamięć współużytkowana umożliwia wspólne używanie zmiennych przez różne programy.
- **Pojedynczy system zarządzania zadaniami.** Dzięki klastrowemu programowi szeregującemu użytkownik może przedkładać zadanie bez określania komputera, który ma je wykonywać.
- **Pojedynczy interfejs użytkownika.** Wspólny interfejs graficzny obsługuje wszystkich użytkowników, niezależnie od stacji roboczej, za pośrednictwem której posługują się oni klastrem.



Rysunek 18.10. Architektura systemu klastrowego [BUYY99a]

- **Pojedyncza przestrzeń wejścia-wyjścia.** Dowolny węzeł może dysponować zdalnym dostępem do dowolnego urządzenia peryferyjnego lub dysku bez znajomości jego lokalizacji fizycznej.
- **Pojedyncza przestrzeń procesów.** Używany jest jednolity system identyfikowania procesów. Proces realizowany w jednym węźle może się komunikować z dowolnym procesem w węźle odległym.
- **Wprowadzanie punktów kontrolnych.** Polega to na okresowym zapisywaniu stanu procesu i pośrednich wyników obliczeń w celu umożliwienia przywracania stanu po uszkodzeniu.
- **Migracja procesu.** Funkcja ta umożliwia równoważenie obciążenia.

Jak widać na rys. 18.10, klaster zawiera również narzędzia programowe umożliwiające efektywne wykonywanie programów, które mogą być realizowane równolegle.

Klastry a SMP

Zarówno klastry, jak i wieloprocesory symetryczne wykorzystują wiele procesorów do obsługi najbardziej wymagających programów aplikacyjnych. Obydwa rozwiązania są dostępne komercyjnie, chociaż SMP były używane znacznie dłużej.

Główną zaletą rozwiązań SMP jest to, że są łatwiejsze do zarządzania i konfigurowania, niż klastry. SMP jest bliższy modelowi jednoprocesorowemu, dla którego zostały napisane niemal wszystkie programy użytkowe. Podstawowa zmiana, jaka jest wymagana przy przechodzeniu od systemu jednoprocesorowego do SMP, dotyczy programu szeregującego. Korzystną stroną SMP jest to, że zwykle zajmuje on mniej przestrzeni fizycznej i pobiera mniej mocy niż porównywalny klaster. Ważne jest wreszcie to, że produkty SMP są ustabilizowane i zadomowione na rynku.

Jednak na dłuższą metę korzyści towarzyszące klastrom doprowadzą prawdopodobnie do ich dominacji na rynku serwerów o najwyższej wydajności. Klastry znacznie przewyższają SMP pod względem przyrostowej i względnej skalowalności. Są również korzystniejsze, jeśli chodzi o dostępność, ponieważ wszystkie składniki systemu mogą być poddane redundancji.

18.5. Niejednorodny dostęp do pamięci

W kategorii produktów komercyjnych dwoma powszechnie stosowanymi rozwiązaniami wieloprocesorowymi są SMP i klastry. Od kilku lat przedmiotem badań było inne rozwiązanie, znane jako *systemy oparte na niejednorodnym dostępie do pamięci* (*nonuniform memory access* – NUMA). Obecnie są już dostępne komercyjne produkty NUMA.

Zanim się zagłębimy w to zagadnienie, powinniśmy zdefiniować kilka wyrażeń często spotykanych w literaturze poświęconej NUMA.

- **Jednorodny dostęp do pamięci** (*uniform memory access* – **UMA**). Wszystkie procesory mają dostęp do wszystkich części pamięci głównej przy użyciu operacji ładowania i zapisu. Czas dostępu procesora do wszystkich rejonów pamięci jest taki sam. Czas dostępu doświadczany przez różne procesory jest taki sam. Organizacja SMP przeanalizowana w podrozdz. 18.2 i 18.3 jest kwalifikowana jako UMA.
- **Niejednorodny dostęp do pamięci (NUMA).** Wszystkie procesory mają dostęp do wszystkich części pamięci głównej przy użyciu operacji ładowania i zapisu. Czas dostępu procesora do pamięci zależy od rejonu, którego ten dostęp dotyczy. To ostatnie zdanie dotyczy wszystkich procesorów; jednak dla różnych procesorów to, które rejony pamięci są wolniejsze, które zaś szybsze, jest odmienne.
- **Systemy NUMA o spójnych pamięciach podręcznych** (*cache-coherent nonuniform memory access* – **CC-NUMA**). Systemy NUMA, w których jest zachowana spójność pamięci podręcznych wśród różnych procesorów.

System NUMA pozbawiony spójności pamięci podręcznych jest w mniejszym lub większym stopniu równoważny klastrowi. Produktami komercyjnymi, które ostatnio przyciągnęły wiele uwagi, są systemy CC-NUMA, wyraźnie odmienne zarówno od SMP, jak i od klastrów. Zwykle – lecz niestety nie zawsze – systemy takie są określane w literaturze komercyjnej jako systemy CC-NUMA. Ten podrozdział jest poświęcony wyłącznie systemom CC-NUMA.

Uzasadnienie

W przypadku SMP istnieje praktyczne ograniczenie liczby procesorów, które mogą być użyte. Efektywny układ pamięci podręcznych ogranicza ruch na magistrali między dowolnym procesorem a pamięcią główną. Gdy jednak wzrasta liczba procesorów, wzrasta również ruch na magistrali. Magistrala służy również do wymiany sygnałów zapewniających spójność pamięci podręcznych, co jeszcze bardziej zwiększa jej obciążenie. W pewnym momencie magistrala staje się wąskim gardłem ograniczającym wydajność. Pogorszenie wydajności wydaje się ograniczać liczbę procesorów w konfiguracji SMP do mniej więcej 16-64 procesorów. Na przykład Power Challenge SMP firmy Silicon Graphics jest ograniczony do 64 procesorów R10000 w jednym systemie; powyżej tej liczby wydajność pogarsza się w sposób istotny.

Ograniczenie liczby procesorów w SMP to jeden z głównych motywów, które spowodowały powstanie i rozwój systemów klastrowych. Jednak w klastrze każdy węzeł dysponuje własną, „prywatną" pamięcią główną; programy aplikacyjne nie widzą jednej pamięci globalnej. W rezultacie spójność jest utrzymywana za sprawą oprogramowania, nie zaś sprzętu. Taka ziarnistość pamięci odbija się na wydajności, aby więc ją utrzymać na maksymalnym poziomie, oprogramowanie musi być dostosowane do takiego właśnie środowiska. Jednym z rozwiązań umożliwiających osiągnięcie wieloprzetwarzania na wielką skalę przy zachowaniu właściwości SMP jest NUMA. Na przykład system Origin NUMA Silicon Graphics został zaprojektowany tak, aby obsługiwał do 1024 procesorów MIPS R10000 [WHIT97], a system Sequent NUMA-Q ma obsługiwać do 252 procesorów Pentium II [LOVE96].

Celem osiąganym dzięki NUMA jest zachowanie przejrzystej pamięci obejmującej cały system przy wielkiej liczbie węzłów procesorowych, z których każdy dysponuje własną magistralą lub innym systemem połączeń wewnętrznych.

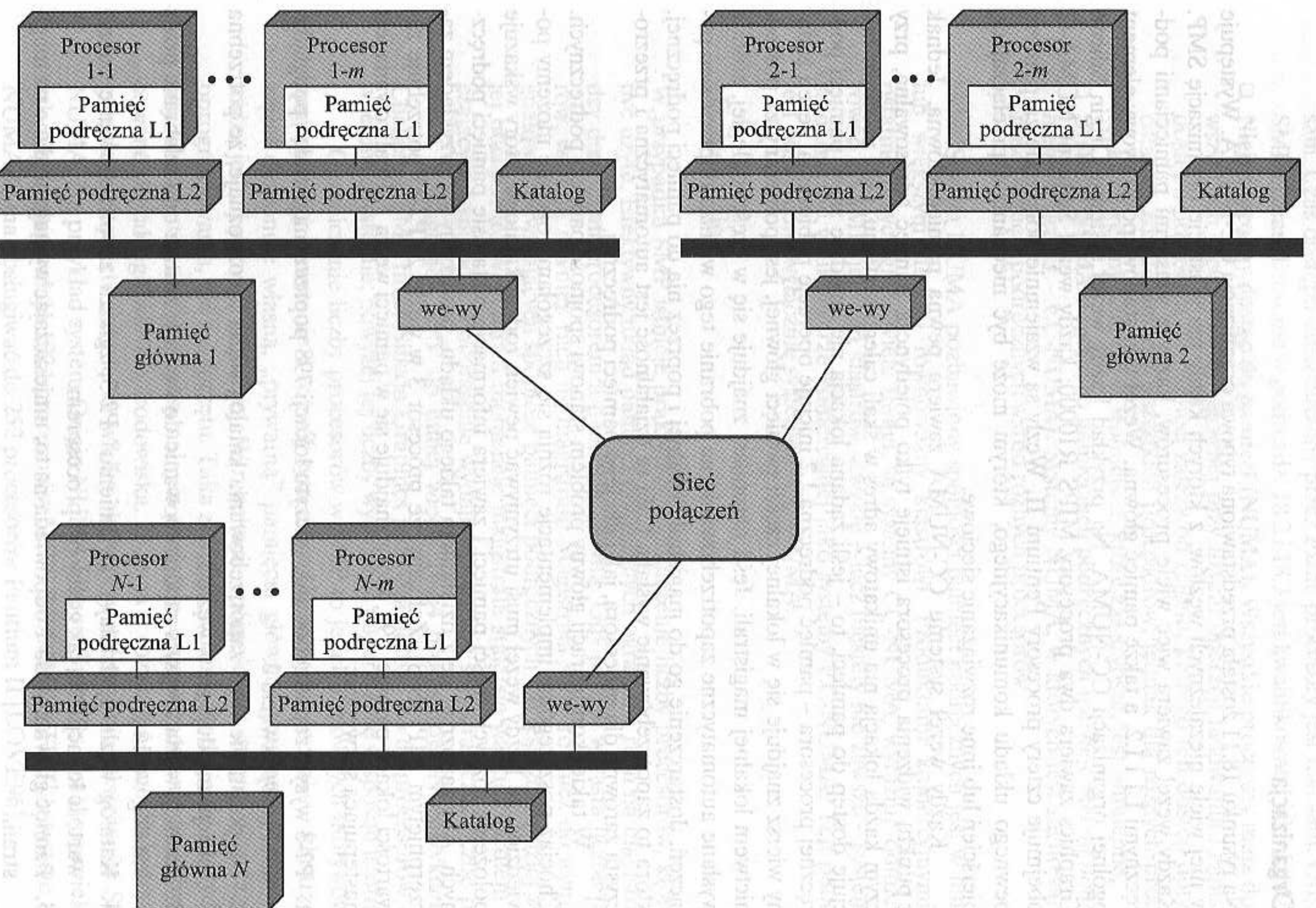
Organizacja

Na rysunku 18.11 została przedstawiona typowa organizacja CC-NUMA. Występuje w niej wiele niezależnych węzłów, z których każdy ma w istocie organizację SMP. Każdy węzeł zawiera więc wiele procesorów, każdy z własnymi pamięciami podręcznymi L1 i L2, a także pamięć główną. Węzeł taki stanowi podstawowy element ogólnej organizacji CC-NUMA. Na przykład każdy węzeł systemu Origin Silicon Graphics zawiera dwa procesory MIPS R10000; każdy węzeł Sequent NUMA-Q obejmuje cztery procesory Pentium II. Węzły są wzajemnie połączone za pomocą pewnego układu komunikacyjnego, którym może być mechanizm przełączający, pierścień lub inne rozwiązanie sieciowe.

Każdy węzeł systemu CC-NUMA zawiera pewną pamięć główną. Jednak z punktu widzenia procesora istnieje tylko pojedyncza pamięć adresowalna, przy czym każda lokacja ma unikatowy adres w skali całego systemu. Gdy procesor inicjuje dostęp do pamięci, to – jeśli żądana lokacja nie znajduje się w pamięci podręcznej procesora – pamięć podręczna L2 inicjuje operację pobierania. Jeżeli żądany wiersz znajduje się w lokalnej części pamięci głównej, jest pobierany za pośrednictwem lokalnej magistrali. Jeśli natomiast znajduje się w części odległej, zostaje wysłane automatyczne zapotrzebowanie na pobranie tego wiersza poprzez sieć połączeń, dostarczenie go do magistrali lokalnej i poprzez nią do pamięci podręcznej, która to zapotrzebowanie wysłała. Cała ta działalność jest automatyczna i przezroczysta zarówno dla procesora, jak i dla jego pamięci podręcznej.

W takiej konfiguracji główny problem stanowi spójność pamięci podręcznych. Chociaż poszczególne implementacje różnią się szczegółami, ogólnie możemy powiedzieć, że każdy węzeł musi utrzymywać pewien rodzaj katalogu, który wskazuje położenie różnych części pamięci i zawiera informacje o statusie pamięci podręcznych. Aby zapoznać się z działaniem takiego układu, posłużymy się przykładem zaczerpniętym z [PFIS98]. Załóżmy, że procesor 3 w węźle 2 (P2-3) potrzebuje zawartości lokacji pamięci 798, która znajduje się w pamięci węzła 1. Realizowana jest następująca sekwencja:

1. P2-3 wysyła zapotrzebowanie odczytu lokacji 798 poprzez magistralę „podglądania" (*snopy*) węzła 2.
2. Na podstawie tego zapotrzebowania katalog węzła 2 rozpoznaje, że potrzebna lokacja znajduje się w węźle 1.
3. Katalog węzła 2 wysyła zapotrzebowanie do węzła 1; zostaje ono odebrane przez katalog węzła 1.
4. Katalog węzła 1, działając „w imieniu" P2-3, zgłasza zapotrzebowanie na zawartość lokacji 798, jak gdyby był procesorem.
5. Pamięć główna węzła 1 odpowiada na to, umieszczając wymagane dane na magistrali.



Rysunek 18.11. Organizacja CC-NUMA

6. Katalog węzła 1 odbiera dane z magistrali.
7. Żądana wartość jest przekazywana do katalogu węzła 2.
8. Katalog węzła 2 umieszcza dane na magistrali węzła 2, działając „w imieniu" pamięci, w której dane te były zawarte.
9. Wartość ta zostaje odebrana, umieszczona w pamięci podręcznej P2-3 i przekazana do tegoż procesora.

Powyższa sekwencja wyjaśnia, w jaki sposób dane są odczytywane w odległej pamięci przy użyciu mechanizmów sprzętowych, dzięki którym transakcja ta jest dla procesora przezroczysta. Niezbędnym uzupełnieniem tych mechanizmów jest jakaś postać protokołu spójności pamięci podręcznych. W różnych systemach spójność tę osiąga się w różny sposób. Tutaj poczynimy jedynie kilka uwag ogólnych. Po pierwsze, w ramach powyższej sekwencji w katalogu węzła 1 jest utrzymywany zapis, że pewna odległa pamięć podręczna ma kopię wiersza z zawartością lokacji 798. Następnie musi istnieć jakiś protokół współpracy wykorzystywany przy modyfikowaniu zawartości pamięci podręcznych. Jeśli na przykład w danej pamięci podręcznej jest dokonywana modyfikacja, fakt ten musi być rozgłoszony do pozostałych węzłów. Katalogi poszczególnych węzłów, które otrzymują taki komunikat, mogą następnie stwierdzić, czy jakakolwiek lokalna pamięć podręczna zawiera dany wiersz, a jeśli tak, powodują jego usunięcie. Jeśli rzeczywista lokacja pamięci znajduje się w węźle odbierającym powiadomienie, to katalog tego węzła musi zachowywać wpis wskazujący, że ten wiersz pamięci jest nieważny; trwa to do czasu dokonania zapisu opóźnionego (*write back*). Jeśli inny procesor (lokalny lub odległy) zgłosi zapotrzebowanie na ten wiersz, to lokalny katalog wymusza zapis opóźniony aktualizujący pamięć, zanim przekaże żądaną wartość.

Za i przeciw rozwiązaniu NUMA

Główną zaletą systemu CC-NUMA jest to, że może on zapewnić efektywniejszą wydajność na wyższych poziomach paralelizmu niż SMP, nie wymagając przy tym poważnych zmian oprogramowania. Przy wielu węzłach NUMA ruch na magistralach poszczególnych węzłów ogranicza się do popytu, który ta magistrala jest w stanie zaspokoić. Jeśli jednak dostępy do pamięci dotyczą węzłów odległych, wydajność zaczyna spadać. Są powody, aby sądzić, że temu spadkowi będzie można zapobiec. Po pierwsze, użycie pamięci podręcznych L1 i L2 ma na celu ograniczenie wszelkich dostępów do pamięci, łącznie z odległymi. Jeśli większość oprogramowania cechuje znaczny stopień lokalności czasowej, to dostępy do pamięci odległych nie będą nadmiernie częste. Po drugie, jeśli oprogramowanie wyróżnia się znacznym stopniem lokalności przestrzennej i jeśli jest wykorzystywana pamięć wirtualna, to dane potrzebne aplikacjom będą rezydowały na ograniczonej liczbie często używanych stron, które mogą być na wstępie załadowane do pamięci lokalnej w stosunku do bieżącej aplikacji. Projektanci Sequent informują, że w reprezentatywnych aplikacjach występuje taka właśnie lokalność przestrzenna [LOVE96]. Wreszcie układ pamięci wirtualnej może być wzbogacony drogą włączenia do systemu operacyjnego

mechanizmu migracji stron, który przenosiłby stronę pamięci wirtualnej do tego węzła, który jej często używa; projektanci Silicon Graphics donoszą o powodzeniu takiego rozwiązania [WHIT97].

Rozwiązanie CC-NUMA ma również swoje wady. Dwie z nich zostały przeanalizowane szczegółowo w [PFIS98]. Po pierwsze, CC-NUMA nie wykazuje takiej przezroczystości jak SMP; aby przenieść system operacyjny i aplikacje z SMP do CC-NUMA, będą wymagane zmiany. Obejmują one alokację stron (o której już była mowa), alokację procesów i równoważenie obciążenia przez system operacyjny. Drugim problemem jest dostępność. Jest to zagadnienie złożone, zależne od dokładnej implementacji systemu CC-NUMA; zainteresowany czytelnik może sięgnąć do [PFIS98].

18.6. Obliczenia wektorowe

Chociaż wydajność dużych komputerów o ogólnym przeznaczeniu wciąż wzrasta, nadal istnieją zastosowania leżące poza zasięgiem współczesnych dużych komputerów. Potrzebne są komputery rozwiązujące problemy matematyczne w rzeczywistych procesach, takich jakie występują w aerodynamice, sejsmologii, meteorologii oraz w fizyce atomowej, jądrowej i plazmowej. Problemy te charakteryzują się zwykle potrzebą wysokiej dokładności i programem, który powtarzalnie wykonuje zmiennopozycyjne operacje arytmetyczne na dużych tablicach liczb.

Większość z tych problemów może być zaliczona do kategorii znanej jako *symulacja pola ciągłego* (*continuous-field simulation*). W istocie, sytuacja fizyczna może być opisana przez powierzchnię lub obszar w trzech wymiarach (np. przepływ powietrza w sąsiedztwie powierzchni rakiety). Powierzchnia ta jest aproksymowana przez siatkę punktów. Zbiór równań różniczkowych określa fizyczne zachowanie powierzchni w każdym punkcie. Równania są reprezentowane jako tablice wartości i współczynników, a rozwiązanie polega na powtarzaniu operacji arytmetycznych na tablicach danych.

Aby umożliwić rozwiązywanie tego rodzaju problemów, opracowano superkomputery. Maszyny te mogą wykonywać setki milionów operacji zmiennopozycyjnych na sekundę, a ich koszt waha się od 10 do 15 milionów dolarów. W przeciwieństwie do dużych komputerów, które zostały zaprojektowane pod kątem wieloprogramowania i intensywnego używania wejścia-wyjścia, superkomputery są optymalizowane pod kątem wyżej opisanych obliczeń numerycznych.

Zakres stosowania superkomputerów jest ograniczony, a ze względu na cenę ograniczony jest również ich rynek. Stosunkowo niewiele tych maszyn pracuje, głównie w centrach badawczych i w niektórych agencjach rządowych wykonujących prace badawcze i inżynierskie. Podobnie jak w innych obszarach techniki komputerowej, także w przypadku superkomputerów występuje ciągłe zapotrzebowanie na zwiększanie ich wydajności. Wobec tego ewolucja technologii i wydajności superkomputerów trwa nadal.

Istnieje inny rodzaj systemu, który został zaprojektowany w celu zaspokojenia zapotrzebowania na obliczenia wektorowe, nazwany *procesorem tablicowym*. Chociaż superkomputery są optymalizowane pod kątem obliczeń wektorowych, pozostają komputerami o ogólnym przeznaczeniu, zdolnymi do przetwarzania skalarnego i ogólnego przetwarzania danych. Procesory tablicowe nie wykonują przetwarzania skalarnego; są skonfigurowane jako urządzenia peryferyjne wykorzystywane przez użytkowników dużych komputerów i minikomputerów w celu wykonywania wektorowych fragmentów programu.

Sposoby rozwiązywania obliczeń wektorowych

Kluczem do projektowania superkomputera lub procesora tablicowego jest uprzytomnienie sobie, że głównym zadaniem jest wykonywanie operacji arytmetycznych na tablicach lub wektorach liczb zmiennopozycyjnych. W komputerze o ogólnym przeznaczeniu wymaga to iteracji każdego elementu tablicy. Rozważmy na przykład dwa wektory (tablice jednowymiarowe) liczb, A i B. Chcielibyśmy je dodać i umieścić wynik w wektorze C. W przykładzie z rys. 18.12 wymaga to wykonania sześciu oddzielnych dodawań. Jak moglibyśmy przyspieszyć to obliczenie? Odpowiedzią jest wprowadzenie pewnej postaci paralelizmu.

Stosowano kilka rozwiązań w celu wprowadzenia paralelizmu do obliczeń wektorowych. Zilustrujemy to na przykładzie. Rozważmy mnożenie wektorowe $C = A \times B$, gdzie A, B i C są macierzami $N \times N$. Elementy macierzy C są równe:

$$c_{i,j} = \sum_{k=1}^{N} a_{i,k} \cdot b_{k,j}$$

gdzie A, B i C mają elementy odpowiednio $a_{i,j}$, $b_{i,j}$ i $c_{i,j}$. Na rysunku 18.13a jest pokazany program w języku Fortran realizujący to obliczenie, który może być wykonywany na zwykłym procesorze skalarnym.

Jednym z rozwiązań poprawiających wydajność może być *przetwarzanie wektorowe*. Zakłada się wtedy, że jest możliwe operowanie na jednowymiarowych wektorach danych. Na rysunku 18.13b jest pokazany program w Fortranie z nowym rozkazem, który umożliwia określanie obliczeń wektorowych. Notacja ($J = 1, N$) wskazuje, że operacje na wszystkich elementach J w danym przedziale powinny być przeprowadzone jako jedna operacja. Pokrótce omówimy sposób, w jaki można to osiągnąć.

$$\begin{bmatrix} 1{,}5 \\ 7{,}1 \\ 6{,}9 \\ 100{,}5 \\ 0 \\ 59{,}7 \end{bmatrix} + \begin{bmatrix} 2{,}0 \\ 39{,}7 \\ 1000{,}003 \\ 11 \\ 21{,}1 \\ 19{,}7 \end{bmatrix} + \begin{bmatrix} 3{,}5 \\ 46{,}8 \\ 1006{,}903 \\ 11{,}5 \\ 21{,}1 \\ 79{,}4 \end{bmatrix}$$

$$A \quad + \quad B \quad + \quad C$$

Rysunek 18.12. Przykład dodawania wektorowego

```
     DO 100 I = 1, N
     DO 100 J = 1, N
     C(I, J) = 0,0
     DO 100 K = 1, N
     C(I, J) = C(I, J) + A(I, K) + B(K, J)
100  CONTINUE
```

(a) Przetwarzanie skalarne

```
     DO 100 I = 1, N
     C(I, J) = 0,0 (J = 1, N)
     DO 100 K = 1, N
     C(I, J) = C(I, J) + A(I, K) + B(K, J) (J = 1, N)
100  CONTINUE
```

(b) Przetwarzanie wektorowe

```
     DO 50 J = 1, N - 1
     FORK 100
50   CONTINUE
     J = N
100  DO 200 I = 1, N
     C(I, J) = 0,0
     DO 200 K = 1, N
     C(I, J) = C(I, J) + A(I, K) + B(K, J)
200  CONTINUE
```

(c) Przetwarzanie równoległe

Rysunek 18.13. Mnożenie macierzowe ($C = A \times B$)

Program z rysunku 18.13b wskazuje, że wszystkie elementy i-tego wiersza macierzy powinny być obliczane równolegle. Każdy element wiersza jest wynikiem sumowania; sumowania te (po k) są wykonywane szeregowo, a nie równolegle. Mimo to potrzeba tylko N^2 mnożeń wektorowych przy zastosowaniu tego algorytmu w porównaniu z N^3 mnożeń skalarnych przy zastosowaniu algorytmu skalarnego.

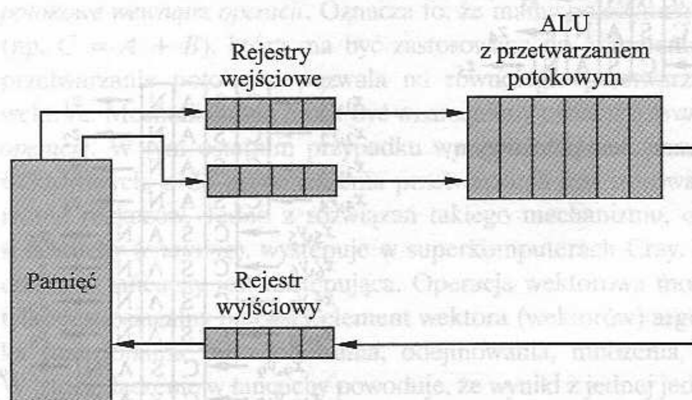
Inne rozwiązanie, *przetwarzanie równoległe*, jest zilustrowane na rys. 18.13c. Zakłada się tutaj, że mamy N niezależnych procesorów, które mogą działać równolegle. W celu efektywnego wykorzystania procesorów musimy jakoś rozdzielić obliczenia na procesory. Używane są dwie funkcje pierwotne. Funkcja FORK n (rozwidlenie n) powoduje zapoczątkowanie niezależnego procesu w lokacji n. W tym samym czasie jest kontynuowane wykonywanie oryginalnego procesu począwszy od rozkazu następującego bezpośrednio po FORK. Każde wykonanie funkcji FORK powoduje powstanie nowego procesu. Rozkaz JOIN (połączenie) jest w zasadzie odwróceniem FORK. Instrukcja JOIN N powoduje, że N niezależnych procesów ulega połączeniu w jeden, który kontynuuje wykonywanie począwszy od rozkazu następującego po JOIN. System operacyjny musi koordynować to połączenie, dzięki czemu wykonywanie nie jest kontynuowane, dopóki wszystkie N procesów nie dotrze do rozkazu JOIN.

Program pokazany na rys. 18.13c odzwierciedla zachowanie programu przetwarzania wektorowego. W przypadku programu przetwarzania równoległego każda kolumna macierzy C jest obliczana za pomocą oddzielnego procesu. Elementy danego wiersza macierzy C są więc obliczane równolegle.

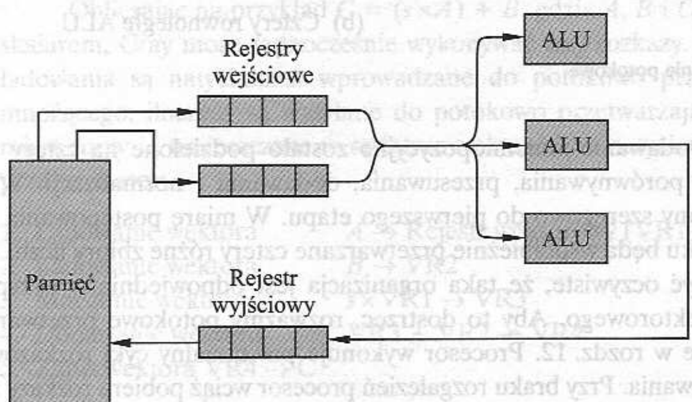
W dotychczasowej dyskusji przedstawiliśmy rozwiązania obliczeń wektorowych w ujęciu logicznym lub architektonicznym. Zajmijmy się teraz rodzajami organizacji procesora, które mogą posłużyć do wdrożenia tych rozwiązań. Próbowano w przeszłości i nadal próbuje się wielu różnych rozwiązań organizacyjnych. Występują trzy główne kategorie:

- ALU z przetwarzaniem potokowym;
- równoległe ALU;
- równoległe procesory.

Na rysunku 18.14 są pokazane dwa pierwsze z tych rozwiązań. Przetwarzanie potokowe przedyskutowaliśmy już w rozdz. 12. Tutaj poszerzymy tę koncepcję o funkcjonowanie ALU. Ponieważ operacje zmiennopozycyjne są raczej złożone, istnieje możliwość dekompozycji tych operacji na etapy, dzięki czemu różne etapy mogą być realizowane współbieżnie na różnych zbiorach danych. Jest to zilustrowane

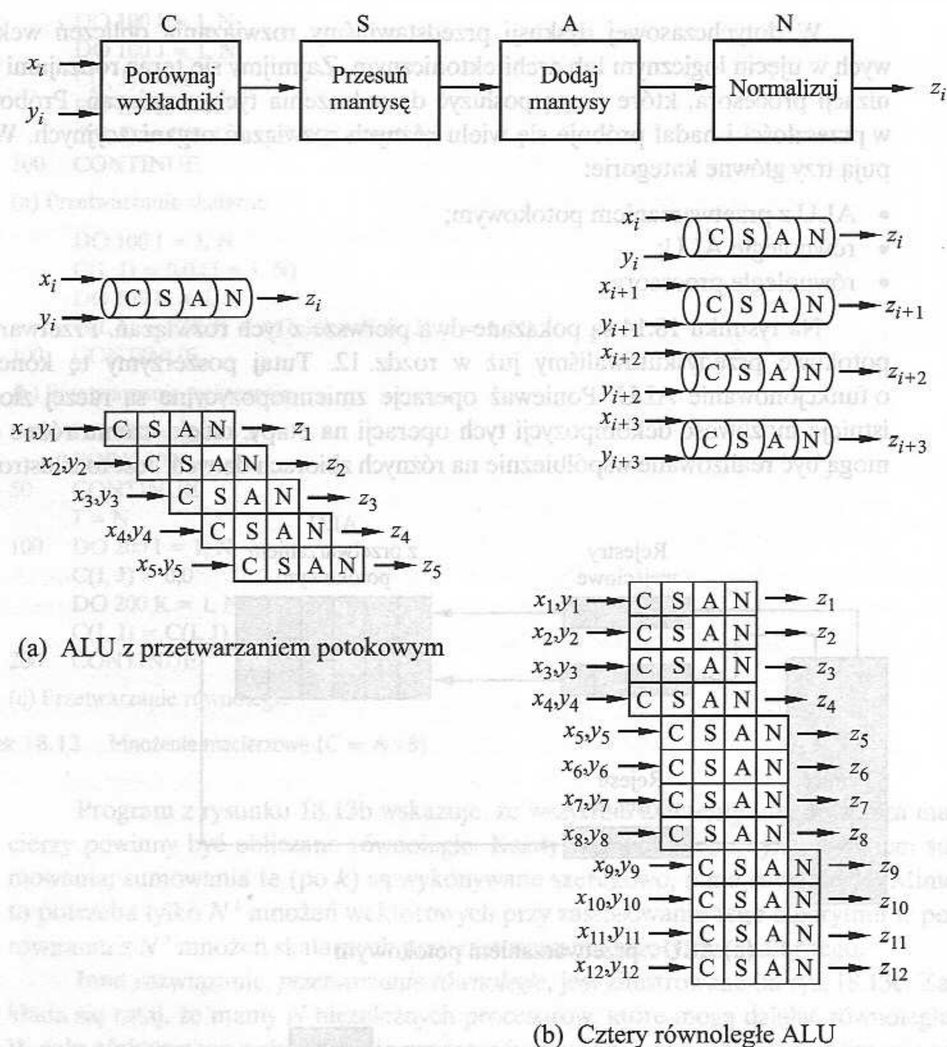


(a) ALU z przetwarzaniem potokowym



(b) Równoległe ALU

Rysunek 18.14. Różne rozwiązania obliczeń wektorowych



Rysunek 18.15. Przetwarzanie potokowe

na rys. 18.15a. Dodawanie zmiennopozycyjne zostało podzielone na cztery etapy (patrz rys. 9.22): porównywania, przesuwania, dodawania i normalizacji. Wektor liczb jest kierowany szeregowo do pierwszego etapu. W miarę postępowania przetwarzania w potoku będą współbieżnie przetwarzane cztery różne zbiory liczb.

Powinno być oczywiste, że taka organizacja jest odpowiednia w przypadku przetwarzania wektorowego. Aby to dostrzec, rozważmy potokowe przetwarzanie rozkazów opisane w rozdz. 12. Procesor wykonuje powtarzalny cykl rozkazów pobierania i wykonywania. Przy braku rozgałęzień procesor wciąż pobiera rozkazy z szeregowo położonych lokacji. W konsekwencji potok jest wypełniony i osiąga się oszczędność czasu. Analogicznie, wykorzystując ALU z przetwarzaniem potokowym, umożliwia się zaoszczędzenie czasu tylko wtedy, kiedy do ALU są dostarczane

strumieniem dane położone w kolejnych lokacjach. Prosta, izolowana operacja zmiennopozycyjna nie ulega przyspieszeniu w wyniku stosowania przetwarzania potokowego. Osiąga się przyspieszenie, gdy do ALU jest doprowadzany wektor argumentów. Jednostka sterująca taktuje przechodzenie danych przez jednostkę ALU, aż do czasu przetworzenia całego wektora.

Można jeszcze bardziej udoskonalić funkcjonowanie potoku, jeśli elementy wektora będą dostępne w rejestrach zamiast w pamięci głównej. W rzeczywistości to właśnie zasugerowano na rys. 18.14a. Elementy każdego wektora argumentów są w postaci bloku ładowane do *rejestru wektorowego*, który jest po prostu dużym bankiem identycznych rejestrów. Wynik jest również umieszczany w rejestrze wektorowym. Wobec tego większość operacji obejmuje tylko użycie rejestrów i tylko operacje ładowania oraz zapisu na początku i na końcu operacji wektorowej wymagają dostępu do pamięci.

Mechanizm zilustrowany na rys. 18.15 może być określony jako *przetwarzanie potokowe wewnątrz operacji*. Oznacza to, że mamy pojedynczą operację arytmetyczną (np. $C = A + B$), która ma być zastosowana do argumentów wektora, natomiast przetwarzanie potokowe pozwala na równoległe przetwarzanie wielu elementów wektora. Mechanizm ten może być wzmocniony przez *przetwarzanie potokowe poprzez operacje*. W tym ostatnim przypadku występuje sekwencja arytmetycznych operacji wektorowych, a do przyspieszenia przetwarzania jest stosowane potokowe przetwarzanie rozkazów. Jedno z rozwiązań takiego mechanizmu, określane jako **łączenie w łańcuchy** (*chaining*), występuje w superkomputerach Cray. Podstawowa reguła łączenia w łańcuchy jest następująca. Operacja wektorowa może być rozpoczęta, gdy tylko jest osiągalny pierwszy element wektora (wektorów) argumentów i gdy jednostka funkcjonalna (np. dodawania, odejmowania, mnożenia, dzielenia) jest wolna. W istocie łączenie w łańcuchy powoduje, że wyniki z jednej jednostki funkcjonalnej są natychmiast kierowane do następnej itd. Jeśli używane są rejestry wektorowe, to wyniki pośrednie nie muszą być zapisywane w pamięci i mogą być używane, zanim jeszcze zakończy się operacja wektorowa, która była ich źródłem.

Obliczając na przykład $C = (s \times A) + B$, gdzie A, B i C są wektorami, a s jest skalarem, Cray może jednocześnie wykonywać trzy rozkazy. Elementy pobrane do ładowania są natychmiast wprowadzane do potokowo przetwarzającego układu mnożącego, iloczyny są wysyłane do potokowo przetwarzającego sumatora, natomiast sumy są umieszczane w rejestrze wektorowym natychmiast po obliczeniu ich przez sumator:

1. Ładowanie wektora $A \to$ Rejestr wektorowy (VR1)
2. Ładowanie wektora $B \to$ VR2
3. Mnożenie wektora $s \times$ VR1 $\to$ VR3
4. Dodawanie wektora VR3 + VR2 $\to$ VR4
5. Zapis wektora VR4 $\to$ C

Rozkazy 2 i 3 mogą być łączone w łańcuch (przetwarzane potokowo), ponieważ dotyczą one innych lokacji w pamięci i rejestrach. Rozkaz 4 potrzebuje wyników rozkazów 2 i 3, jednak również może być wraz z nimi łączony w łańcuch. Gdy tylko

są osiągalne pierwsze elementy rejestrów wektorowych 2 i 3, może się rozpocząć operacja należąca do rozkazu 4.

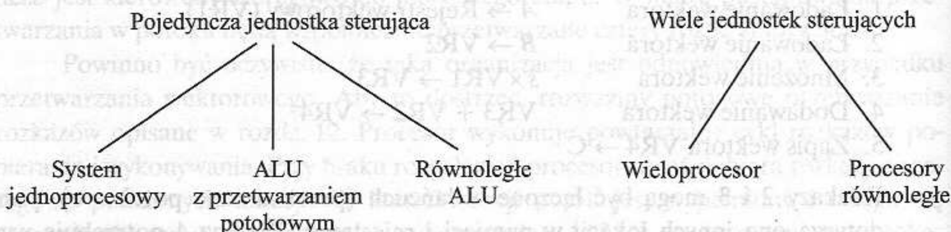
Innym sposobem przetwarzania wektorowego jest użycie wielu ALU w jednym procesorze pod kontrolą jednej jednostki sterującej. W tym przypadku jednostka sterująca rozprowadza dane do wszystkich ALU, dzięki czemu mogą one pracować równolegle. Jest również możliwe użycie przetwarzania potokowego w każdej z równoległych ALU. Jest to zilustrowane na rys. 18.15b. W tym przykładzie równolegle funkcjonują cztery ALU.

Podobnie jak organizacja potokowa, także zastosowanie równoległych ALU jest odpowiednie do przetwarzania wektorowego. Jednostka sterująca rozprowadza elementy wektora do wszystkich ALU w sposób cykliczny, aż do przetworzenia wszystkich elementów. Ten rodzaj organizacji jest bardziej złożony niż procesor z jedną ALU.

Przetwarzanie wektorowe może być wreszcie realizowane za pomocą wielu równolegle przetwarzających procesorów. W tym przypadku konieczne jest rozbicie zadania na wiele procesów, które mogą być wykonywane równolegle. Organizacja taka jest efektywna tylko wtedy, kiedy jest dostępne oprogramowanie i sprzęt umożliwiające skuteczną koordynację równoległych procesorów.

Możemy teraz poszerzyć naszą taksonomię przedstawioną w podrozdz. 18.1 w celu uwzględnienia tych nowych struktur, co widać na rys. 16.11. Organizacje komputerów mogą być rozróżnione na podstawie obecności jednej lub wielu jednostek sterujących. Wiele jednostek sterujących implikuje zastosowanie wielu procesorów. Zgodnie z naszymi poprzednimi rozważaniami, jeśli wiele procesorów może współpracować przy wykonywaniu danego zadania, są one określane jako *procesory równoległe*.

Należy wystrzegać się pewnej niefortunnej terminologii spotykanej w literaturze. Termin *procesor wektorowy* jest często zrównywany z ALU o organizacji potokowej, chociaż równoległa organizacja ALU jest również przeznaczona do przetwarzania wektorowego i, jak mówiliśmy, organizacja procesorów równoległych również może służyć przetwarzaniu wektorowemu. *Przetwarzanie tablicowe* jest czasem wiązane z równoległymi ALU, chociaż każda z tych trzech organizacji również jest optymalizowana pod kątem przetwarzania tablic. Jest jeszcze gorzej; termin *procesor tablicowy* zwykle odnosi się do procesora pomocniczego, który jest dołączony do procesora o ogólnym przeznaczeniu i jest używany do prowadzenia obliczeń wektorowych. W procesorach tablicowych mogą być stosowane ALU o organizacji potokowej lub równoległej.



Rysunek 18.16. Taksonomia organizacji komputerów

Obecnie na rynku dominuje organizacja z ALU przetwarzającą potokowo. Systemy z przetwarzaniem potokowym są mniej złożone niż dwa pozostałe rozwiązania. Projektowanie jednostek sterujących i systemów operacyjnych zostało opanowane na tyle dobrze, żeby uzyskać efektywne przydzielanie zasobów i wysoką wydajność. Pozostałą część tego podrozdziału poświęcimy bardziej szczegółowej analizie tego właśnie rozwiązania na podstawie konkretnego przykładu.

Urządzenie wektorowe IBM 3090

Dobrym przykładem organizacji z ALU przetwarzającą potokowo, przeznaczonej do przetwarzania wektorowego, jest urządzenie wektorowe opracowane przez IBM dla architektury IBM 370 i wdrożone w największych systemach serii 3090 [PADE88, TUCK87]. Urządzenie to jest opcjonalnym dodatkiem do systemu bazowego, jednak jest z nim ściśle zintegrowane. Zastępuje ono rozwiązania przetwarzania wektorowego występujące w superkomputerach, takich jak rodzina Cray.

Urządzenie IBM używa pewnej liczby rejestrów wektorowych. Każdy z nich jest w istocie bankiem rejestrów skalarnych. W celu obliczenia sumy wektorowej $C = A + B$, wektory A i B są ładowane do dwóch rejestrów wektorowych. Dane z tych rejestrów są przeprowadzane przez ALU tak szybko, jak tylko jest to możliwe, wyniki zaś są zapisywane w trzecim rejestrze wektorowym. Nakładanie się obliczeń i ładowanie danych wejściowych do rejestrów w postaci bloku powodują znaczne przyspieszenie w porównaniu ze zwykłą operacją ALU.

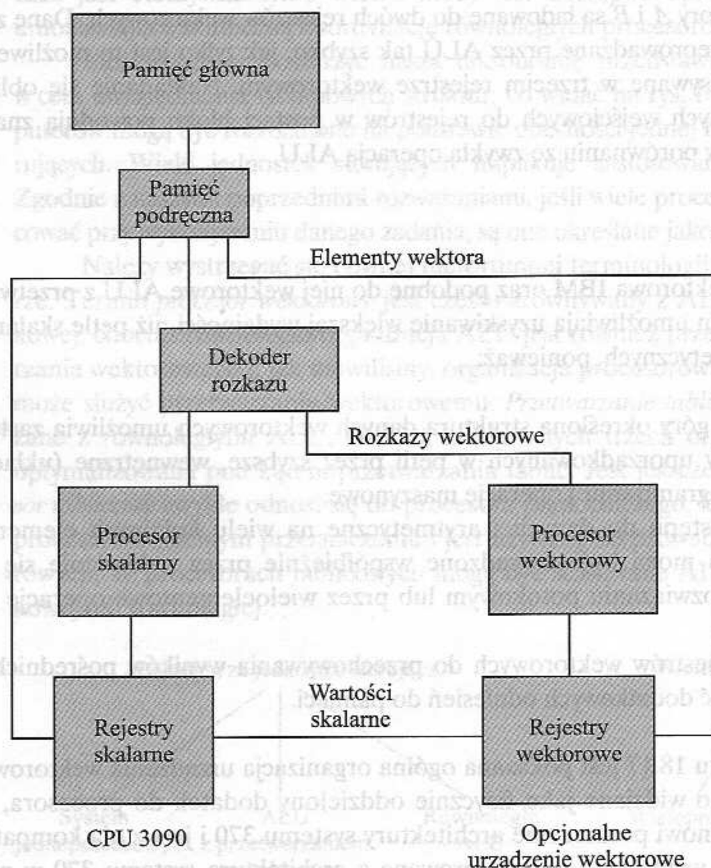
Organizacja

Architektura wektorowa IBM oraz podobne do niej wektorowe ALU z przetwarzaniem potokowym umożliwiają uzyskiwanie większej wydajności niż pętle skalarnych rozkazów arytmetycznych, ponieważ:

- Ustalona i z góry określona struktura danych wektorowych umożliwia zastąpienie rozkazów uporządkowanych w pętli przez szybsze, wewnętrzne (układowe lub mikroprogramowane) operacje maszynowe.
- Operacje dostępu do danych i arytmetyczne na wielu kolejnych elementach wektorowych mogą być prowadzone współbieżnie przez nakładanie się tych operacji w rozwiązaniu potokowym lub przez wieloelementowe operacje równoległe.
- Używanie rejestrów wektorowych do przechowywania wyników pośrednich pozwala uniknąć dodatkowych odniesień do pamięci.

Na rysunku 18.17 jest pokazana ogólna organizacja urządzenia wektorowego. Chociaż jest ono widziane jako fizycznie oddzielony dodatek do procesora, jego architektura stanowi poszerzenie architektury systemu 370 i jest z nią kompatybilna. Urządzenie wektorowe jest integrowane z architekturą systemu 370 w następujący sposób:

- Do wszystkich operacji skalarnych są używane istniejące rozkazy systemu 370.
- Operacje arytmetyczne na indywidualnych elementach wektorów dają dokładnie takie same wyniki, jak odpowiednie rozkazy skalarne systemu 370. Na przykład jedna z decyzji projektowych dotyczyła określenia wyniku zmiennopozycyjnej operacji DIVIDE (dzielenia). Czy wynik powinien być tak samo dokładny, jak wynik zmiennopozycyjnego dzielenia skalarnego, czy też dopuszczalna jest pewna aproksymacja umożliwiająca zwiększenie szybkości, lecz czasem prowadząca do błędu na jednej lub wielu pozycjach bitowych niskiego rzędu? Podjęto decyzję utrzymania całkowitej kompatybilności z architekturą systemu 370 kosztem nieznacznego pogorszenia wydajności.
- Rozkazy wektorowe mogą być przerywane, a ich wykonywanie może być wznawiane od punktu przerwania po podjęciu odpowiednich działań w sposób kompatybilny ze schematem obsługi programu przerwań w systemie 370.
- Wyjątki arytmetyczne są takie same lub stanowią poszerzenie w stosunku do wyjątków związanych ze skalarnymi rozkazami arytmetycznymi systemu 370, mogą też być używane podobne podprogramy porządkujące. W tym celu zastosowano



Rysunek 18.17. IBM 3090 z urządzeniem wektorowym

wskaźnik przerwania wektorowego wskazujący położenie w rejestrze wektorowym, którego dotyczy wyjątek (np. przepełnienie). Po przywróceniu wykonywania rozkazu wektorowego następuje więc dostęp do właściwego miejsca w rejestrze wektorowym.

- Dane wektorowe pozostają w pamięci wirtualnej, przy czym błędy stron są traktowane w sposób standardowy.

Ten poziom integracji jest powodem wielu korzyści. Istniejące systemy operacyjne mogą współpracować z urządzeniem wektorowym po niewielu tylko rozszerzeniach. Istniejące programy aplikacyjne, kompilatory języka i inne oprogramowanie mogą być używane bez zmian. W celu wykorzystania możliwości przetwarzania wektorowego oprogramowanie może też być modyfikowane według życzenia.

Rejestry

Kluczowym zagadnieniem przy projektowaniu urządzenia wektorowego jest to, czy argumenty są umieszczane w rejestrach, czy w pamięci. Organizacja IBM jest określana jako *z rejestru do rejestru*, ponieważ zarówno wejściowe, jak i wyjściowe argumenty wektorowe mogą być umieszczane w rejestrach wektorowych. Rozwiązanie takie zostało również użyte w superkomputerze Cray. Rozwiązaniem alternatywnym, stosowanym w maszynach Control Data, jest uzyskiwanie argumentów bezpośrednio z pamięci. Główną niedogodnością używania rejestrów wektorowych jest to, że muszą one być uwzględniane przez programistę lub przez kompilator. Załóżmy na przykład, że długość rejestrów wektorowych wynosi K, a długość przetwarzanych wektorów jest równa $N > K$. W takim przypadku musi być wykonana pętla wektorowa, podczas której w określonej chwili jest prowadzona operacja na K elementach, a pętla jest powtarzana N/K razy. Główną zaletą stosowania rejestrów wektorowych jest to, że operacje są oderwane od wolniejszej pamięci i zachodzą przede wszystkim w rejestrach.

Przyspieszenie, jakie można osiągnąć dzięki rejestrom, jest zilustrowane na rys. 18.18 [PADE88]. Program Fortran mnoży wektor A przez wektor B w celu uzyskania C, przy czym każdy wektor ma część rzeczywistą (AR, BR, CR) i urojoną (AI, BI, CI). Urządzenie 3090 może dokonywać jednego dostępu do pamięci głównej na jeden cykl procesora lub zegara (może to być zapis albo odczyt). Urządzenie to ma rejestry, które w ciągu cyklu umożliwiają dwa dostępy dla odczytu i jeden dla zapisu. W jednostce arytmetycznej urządzenia powstaje jeden wynik w ciągu jednego cyklu. Załóżmy użycie rozkazów, które umożliwiają określanie dwóch argumentów źródłowych i wyniku[4]. Na części rysunku widać, że w przypadku rozkazów z pamięci do pamięci każda iteracja wymaga łącznie 18 cykli. W przypadku architektury typu

[4] W architekturze 370 jedyne 3-argumentowe rozkazy (rozkazy rejestru i pamięci, RS) określają dwa argumenty w rejestrach i jeden w pamięci. W części tego przykładu zakładamy istnienie rozkazów 3-argumentowych, których wszystkie argumenty znajdują się w pamięci głównej. Czynione to jest do celów porównawczych. W rzeczywistości taki właśnie format rozkazów mógłby być wybrany dla architektury wektorowej.

Program w języku FORTRAN:

```
      DO 100 J = 1, 50
      CR(J) = AR(J) *BR(J) – AI(J)*BI(J)
100   CI(J) = AR(J) *BI(J) – AI(J)*BR(J)
```

Operacja		Cykle
AR(J) *BR(J)	→ T1(J)	3
AI(J) *BI(J)	→ T2(J)	3
T1(J) – T2(J)	→ CR(J)	3
AR(J) *BI(J)	→ T3(J)	3
AI(J) *BR(J)	→ T4(J)	3
T3(J) + T4(J)	→ CI(J)	3
Razem		18

(a) Pamięć-pamięć

Operacja		Cykle
AR(J)	→ V1(J)	1
BR(J)	→ V2(J)	1
V1 (J) *V2(J)	→ V3(J)	1
AI(J)	→ V4(J)	1
BI(J)	→ V5(J)	1
V4 (J) *V5(J)	→ V6(J)	1
V3(J) – V6(J)	→ V7(J)	1
V7(J)	→ CR(J)	1
V1(J) *V5(J)	→ V8(J)	1
V4(J) *V2(J)	→ V9(J)	1
V8(J) + V9(J)	→ V0(J)	1
V0(J)	→ CI(J)	1
Razem		12

(b) Rejestr-rejestr

Operacja		Cykle
AR(J)	→ V1(J)	1
V1(J) *BR(J)	→ V2(J)	1
AI(J)	→ V3(J)	1
V3(J) *BI(J)	→ V4(J)	1
V2(J) – V4(J)	→ V5(J)	1
V5(J)	→ CR(J)	1
V1(J) *BI(J)	→ V6(J)	1
V4(J) *BR(J)	→ V7(J)	1
V6(J) + V7(J)	→ V8(J)	1
V8(J)	→ CI(J)	1
Razem		10

(c) Pamięć-rejestr

Vi – rejestry wektorowe
AR, BR, AI, BI – argumenty w pamięci
Ti – tymczasowe lokacje w pamięci

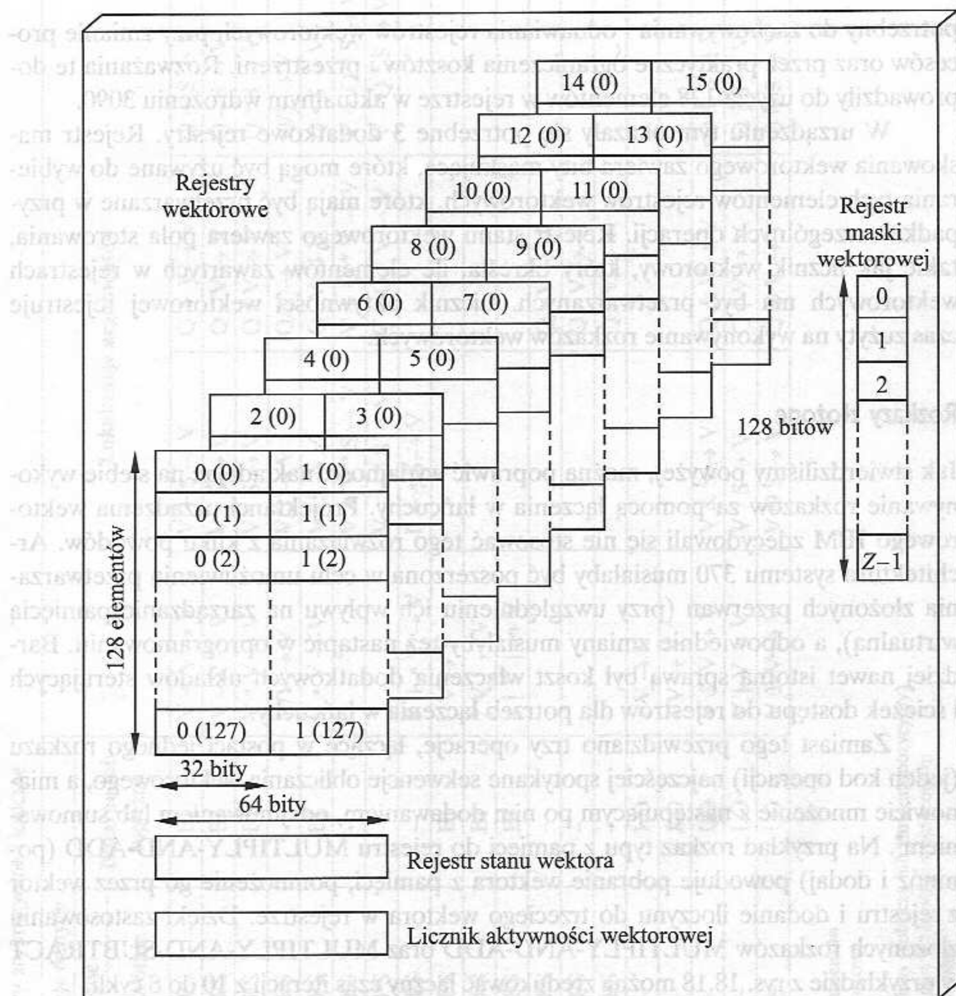
Operacja		Cykle
AR(J)	→ V1(J)	1
V1(J) * BR(J)	→ V2(J)	1
AI(J)	→ V3(J)	1
V2(J) – V3(J) * BI(J)	→ V2(J)	1
V2(J)	→ CR (J)	1
V1(J) * BI(J)	→ V4(J)	1
V4(J) + V3(J) * BR(J)	→ V5(J)	1
V5(J)	→ CI(J)	1
Razem		8

(d) Rozkazy złożone

Rysunek 18.18. Alternatywne programy obliczeń wektorowych

z rejestru do rejestru (część b) czas ten został zredukowany do 12 cykli. Oczywiście w przypadku operacji z rejestru do rejestru wielkości wektorowe muszą być przed obliczaniem załadowane do rejestrów wektorowych i zapisane w pamięci po zakończeniu obliczeń. Przy dużych wektorach ta strata jest stosunkowo niewielka. Na rysunku 16.18c widać, że możliwość określania w jednym rozkazie zarówno argumentów w rejestrach, jak i w pamięci, powoduje zmniejszenie czasu iteracji do 10 cykli. Ten ostatni rodzaj rozkazów został uwzględniony w architekturze wektorowej[5].

[5] Przedyskutowane w dalszym ciągu rozkazy mieszane umożliwiają dalszą redukcję.



Rysunek 18.19. Rejestry urządzenia wektorowego IBM 3090

Na rysunku 18.19 są pokazane rejestry stanowiące część urządzenia wektorowego IBM 3090. Występuje tu 16 32-bitowych rejestrów wektorowych. Rejestry wektorowe mogą być łączone w celu utworzenia 8 rejestrów 64-bitowych. Każdy element rejestru może zawierać liczbę całkowitą lub zmiennopozycyjną. Rejestry wektorowe mogą więc służyć do przechowywania 32- i 64-bitowych wartości całkowitych oraz 32- i 64-bitowych wartości zmiennopozycyjnych.

W tej architekturze każdy rejestr zawiera od 8 do 512 elementów skalarnych. Wyborowi rzeczywistej długości towarzyszy pewien dylemat projektowy. Czas wykonania operacji wektorowej składa się w zasadzie z czasu uruchomienia potoku i zapełnienia rejestru plus jeden cykl na każdy element wektora. Wobec tego używanie dużej liczby elementów rejestrów powoduje zmniejszenie względnego czasu uruchamiania obliczeń. Ta efektywność jest jednak równoważona przez dodatkowy czas

potrzebny do zachowywania i odnawiania rejestrów wektorowych przy zmianie procesów oraz przez praktyczne ograniczenia kosztów i przestrzeni. Rozważania te doprowadziły do użycia 128 elementów w rejestrze w aktualnym wdrożeniu 3090.

W urządzeniu tym okazały się potrzebne 3 dodatkowe rejestry. Rejestr maskowania wektorowego zawiera bity maskujące, które mogą być używane do wybierania tych elementów rejestrów wektorowych, które mają być przetwarzane w przypadku szczególnych operacji. Rejestr stanu wektorowego zawiera pola sterowania, takie jak licznik wektorowy, który określa, ile elementów zawartych w rejestrach wektorowych ma być przetwarzanych. Licznik aktywności wektorowej rejestruje czas zużyty na wykonywanie rozkazów wektorowych.

Rozkazy złożone

Jak stwierdziliśmy powyżej, można poprawić wydajność, nakładając na siebie wykonywanie rozkazów za pomocą łączenia w łańcuchy. Projektanci urządzenia wektorowego IBM zdecydowali się nie stosować tego rozwiązania z kilku powodów. Architektura systemu 370 musiałaby być poszerzona w celu umożliwienia przetwarzania złożonych przerwań (przy uwzględnieniu ich wpływu na zarządzanie pamięcią wirtualną), a odpowiednie zmiany musiałyby też nastąpić w oprogramowaniu. Bardziej nawet istotną sprawą był koszt włączenia dodatkowych układów sterujących i ścieżek dostępu do rejestrów dla potrzeb łączenia w łańcuchy.

Zamiast tego przewidziano trzy operacje, łączące w postaci jednego rozkazu (jeden kod operacji) najczęściej spotykane sekwencje obliczania wektorowego, a mianowicie mnożenie z następującym po nim dodawaniem, odejmowaniem lub sumowaniem*. Na przykład rozkaz typu z pamięci do rejestru MULTIPLY-AND-ADD (pomnóż i dodaj) powoduje pobranie wektora z pamięci, pomnożenie go przez wektor z rejestru i dodanie iloczynu do trzeciego wektora w rejestrze. Dzięki zastosowaniu złożonych rozkazów MULTIPLY-AND-ADD oraz MULTIPLY-AND-SUBTRACT w przykładzie z rys. 18.18 można zredukować łączny czas iteracji z 10 do 8 cykli.

W przeciwieństwie do łączenia w łańcuchy, złożone rozkazy nie wymagają dodatkowych rejestrów do czasowego przechowywania wyników pośrednich i potrzebują o jeden mniej dostęp do rejestru. Rozważmy na przykład następujący łańcuch:

```
A → VR1
VR1 + VR2 → VR1
```

W tym przypadku są wymagane dwa zapisy w rejestrze wektorowym VR1. W architekturze IBM występuje rozkaz typu z pamięci do rejestru ADD (dodaj). Przy użyciu tego rozkazu tylko suma jest umieszczana w VR1. Rozkaz złożony czyni również niepotrzebnym odzwierciedlanie w opisie stanu maszyny współbieżnego wykonywania pewnej liczby rozkazów, co upraszcza zachowywanie i odtwarzanie stanu przez system operacyjny oraz przetwarzanie przerwań.

* Sumowanie jest tu rozumiane jako sumowanie elementów wektora wykonywane na rozkaz ACCUMULATE (*przyp. tłum.*).

Tabela 18.4. Urządzenie wektorowe IBM 3090: rozkazy arytmetyczne i logiczne

Operacja	Rodzaje danych			Lokalizacje argumentów			
	Zmiennopozycyjne		Binarne lub logiczne				
	Długie	Krótkie					
Dodajss	FL	FS	BI	V + V → V	V + S → V	Q + V → V	Q + S → V
Odejmij	FL	FS	BI	V – V → V	V – S → V	Q – V → V	Q – S → V
Pomnóż	FL	FS	BI	V * V → V	V * S → V	Q * V → V	Q * S → V
Podziel	FL	FS	–	V/V → V	V/S → V	Q/V → V	Q/S → V
Porównaj	FL	FS	BI	V • V → V	V · S → V	Q · V → V	Q • S → V
Pomnóż i dodaj	FL	FS	–		V + V * S → V	V + Q * V → V	V + Q * S → V
Pomnóż i odejmij	FL	FS	–		V – V * S → V	V – Q * V → V	V – Q * S → V
Pomnóż i akumuluj	FL	FS	–	P + • V → V	P + • S → V		
Dopełnij	FL	FS	BI	– V → V			
Dodatnia bezwzględna	FL	FS	BI	| V | → V			
Ujemna bezwzględna	FL	FS	BI	– | V | → V			
Maksymalna	FL	FS	–			Q • V → V	
Maksymalna bezwzględna	FL	FS	–			Q • V → V	
Minimalna	FL	FS	–			Q • V → V	
Przesuń logicznie w lewo	–	–	LO	• V → V			
Przesuń logicznie w prawo	–	–	LO	• V → V			
Wykonaj AND	–	–	LO	V & V → V	V & S → V	Q & V → V	Q & S → V
Wykonaj OR	–	–	LO	V | V → V	V | S → V	Q | V → V	Q | S → V
Wykonaj Exclusive-OR	–	–	LO	V ⊕ V → V	V ⊕ S → V	Q ⊕ V → V	Q ⊕ S → V

Objaśnienia:

Rodzaje danych
FL – długa zmiennopozycyjna
FS – krótka zmiennopozycyjna
BI – binarna całkowita
LO – logiczna

Lokalizacje argumentów
V – rejestr wektorowy
S – pamięć
Q – rejestr skalarny (roboczy lub zmiennopozycyjny)
P – sumy cząstkowe w rejestrze wektorowym
• – operacja specjalna

Lista rozkazów

W tabeli 18.4 są podsumowane operacje arytmetyczne i logiczne określone dla architektury wektorowej. Ponadto istnieją rozkazy ładowania z pamięci do rejestru i rozkazy zapisu z rejestru do pamięci. Zauważmy, że wiele z tych rozkazów używa formatu 3-argumentowego. Istnieje także pewna liczba wariantów wielu rozkazów, zależnie od położenia argumentów. Argumentem źródłowym może być zawartość rejestru wektorowego (V), pamięci (S) lub rejestru skalarnego (Q). Miejscem docelowym jest zawsze rejestr wektorowy, z wyjątkiem porównania, którego wynik trafia do rejestru maskowania wektorowego. Przy uwzględnieniu tych wszystkich wariantów łączna liczba kodów operacji (oddzielnych rozkazów) wynosi 171. Jest to raczej duża liczba, jednak nie jest to aż tak kosztowne przy wdrażaniu, jak można by sobie wyobrażać. Gdy maszyna zawiera jednostki arytmetyczne, ścieżki danych dostarczające argumentów z pamięci, rejestry skalarne i rejestry wektorowe związane z potokami wektorowymi, główne koszty zostały już poniesione. Przedstawiona architektura może, przy niewielkim koszcie dodatkowym, dostarczyć bogatego zbioru wariantów wykorzystania tych rejestrów i potoków.

Większość rozkazów zamieszczonych w tablicy 18.4 jest zrozumiała sama przez się. Dwa rozkazy sumowania wymagają dodatkowych wyjaśnień. Operacja akumulowania powoduje zsumowanie elementów pojedynczego wektora (ACCUMULATE) lub elementów iloczynu dwóch wektorów (MULTIPLY-AND-ACCUMULATE). Rozkazy te stanowią interesujący problem projektowy. Chcielibyśmy wykonywać te operacje jak najszybciej, w pełni wykorzystując możliwości potoku ALU. Trudność polega na tym, że suma dwóch liczb wprowadzona do potoku jest osiągalna dopiero po upływie kilku cykli. Wobec tego trzeci element wektora nie może być dodany do sumy dwóch pierwszych elementów, dopóki te elementy nie przejdą przez cały potok. W celu przezwyciężenia tego problemu elementy wektora są dodawane w taki sposób, żeby powstały cztery sumy cząstkowe. W szczególności kolejno dodawane elementy 0, 4, 8, 12, ..., 124 dają sumę cząstkową 0; elementy 1, 5, 9, 13, ..., 125 sumę cząstkową 1; elementy 2, 6, 10, 14, ..., 126 sumę cząstkową 2 oraz elementy 3, 7, 11, 15, ..., 127 sumę cząstkową 4. Każda z tych sum cząstkowych może przechodzić przez potok z maksymalną szybkością, ponieważ opóźnienie w potoku wynosi około 4 cykli. Do przechowywania sum cząstkowych jest używany oddzielny rejestr wektorowy. Gdy wszystkie elementy oryginalnego wektora zostały przetworzone, sumowane są 4 sumy cząstkowe, dając wynik końcowy. Wydajność tej drugiej fazy nie jest krytyczna, ponieważ obejmuje ona tylko 4 elementy wektorowe.

18.7. Polecana literatura

W [CATA94] dokonano przeglądu podstaw wieloprocesorów i szczegółowo przeanalizowano wieloprocesory SMP oparte na SPARC. SMP omówiono również dość szczegółowo w [STON93] i [HWAN93].

[MILE00] to przegląd algorytmów i metod zapewniania spójności pamięci podręcznych w wieloprocesorach, ze szczególnym zwróceniem uwagi na zagadnienia wydajności.

Inny przegląd problemów związanych ze spójnością pamięci podręcznych w wieloprocesorach to [LILJ93]. W [TOMA93] zamieszczono przedruki wielu kluczowych artykułów na ten temat.

[PFIS98] to podstawowa lektura dla każdego, kto jest zainteresowany klastrami; przedstawiono zagadnienia projektowania sprzętu i oprogramowania, porównując ze sobą klastry z SMP i NUMA; przedstawiono również gruntowny opis techniczny problemów projektowania SMP i NUMA. Dogłębne omówienie klastrów można znaleźć w [BUYY99a] i [BUYY99b]. Przegląd klastrów w ujęciu mniej technicznym, wraz z dobrymi komentarzami na temat różnych produktów komercyjnych znajduje się w [WEYG01].

Wartościowe omówienie obliczeń wektorowych można znaleźć w pracach [STON93] i [HWAN93].

BUYY99a Buyya R.: *High Performance Cluster Computing: Architectures and Systems*. Upper Saddle River, Prentice Hall, 1999.

BUYY99b Buyya R.: *High Performance Cluster Computing: Programming and Applications*. Upper Saddle River, Prentice Hall, 1999.

CATA94 Catanzaro B.: *Multiprocessor System Architectures*. Mountain View, Sunsoft Press, 1994.

HWAN93 Hwang K.: *Advanced Computer Architecture*. New York, McGraw-Hill, 1993.

LILJ93 Lilja D.: „Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons". *ACM Computing Surveys*, September 1993.

MILE00 Milenkovic A.: „Achieving High Performance in Bus-Based Shared Memory Multiprocessors". *IEEE Concurrency*, July-September 2000.

PFIS98 Pfister G.: *In Search of Clusters*. Upper Saddle River, Prentice Hall, 1998.

STON93 Stone H.: *High-Performance Computer Architecture*. Reading, Addison-Wesley, 1993.

TOMA93 Tomasevic M., Milutinovic V.: *The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions*. Los Alamitos, IEEE Computer Society Press, 1993.

WEYG01 Weygant P.: *Clusters for High Availability*. Upper Saddle River, Prentice Hall, 2001.

18.8. Podstawowe terminy, pytania kontrolne i problemy do rozwiązania

Podstawowe terminy i ich angielskie odpowiedniki

Jednorodny dostęp do pamięci (UMA) – *uniform memory access*
Klaster – *cluster*
Niejednorodny dostęp do pamięci (NUMA) – *nonuniform memory access*
Protokół katalogowy – *directory protocol*
Protokół MESI – *MESI protocol*
Protokół podglądania – *snoopy protocol*
Przejmowanie danych – *failover*
Przywracanie – *failback*
Rezerwa bierna – *passive standby*
Rezerwa czynna – *active standby*
Spójność pamięci podręcznych – *cache coherence*
System jednoprocesorowy – *uniprocessor*
Urządzenie wektorowe – *vector facility*
Wieloprocesor – *multiprocessor*
Wieloprocesor symetryczny (SMP) – *symmetric multiprocessor*

Pytania kontrolne

18.1. Wymień i krótko zdefiniuj trzy rodzaje organizacji systemów komputerowych.

18.2. Jakie są podstawowe właściwości SMP?

18.3. Jakie są potencjalne zalety SMP w porównaniu z systemami jednoprocesorowymi?

18.4. Jakie są podstawowe problemy projektowania systemów operacyjnych dla SMP?

18.5. Jaka jest różnica między programowymi a sprzętowymi sposobami zapewniania spójności pamięci podręcznych?

18.6. Jakie jest znaczenie poszczególnych czterech stanów w protokole MESI?

18.7. Jakie są podstawowe korzyści wynikające ze stosowania klastrów?

18.8. Jaka jest różnica między przejmowaniem danych a przywracaniem?

18.9. Jakie są różnice między UMA, NUMA i CC-NUMA?

Problemy do rozwiązania

18.1. Niech α będzie procentem kodu programu, który może być wykonywany równolegle przez n procesorów w systemie komputerowym. Załóżmy, że pozostały kod musi być wykonywany sekwencyjnie przez pojedynczy procesor. Szybkość wykonywania rozkazów przez każdy procesor wynosi x MIPS.*

(a) Wyprowadź wyrażenie na efektywną szybkość wykonywania rozkazów w zależności od n, α i x, gdy program ten będzie realizowany wyłącznie przez ten system.

(b) Jeśli $n = 16$ i $x = 4$ MIPS, określ wartość α, przy której system uzyska wydajność 40 MIPS.

18.2. Do wieloprocesora z 8 procesorami jest dołączonych 20 napędów taśmowych. Do systemu została wprowadzona wielka liczba zadań, z których każde wymaga do ukończenia maksymalnie 4 napędów taśmowych. Załóżmy, że każde zadanie wymaga przez dłuższy czas trzech tylko napędów taśmowych, po czym pod koniec wykonywania przez krótki czas jest potrzebny czwarty napęd. Załóżmy również niekończące się wprowadzanie takich zadań.

(a) Załóżmy, że program szeregujący w systemie operacyjnym nie uruchomi zadania, dopóki nie będą dostępne 4 napędy. Gdy zadanie jest uruchamiane, 4 napędy są mu natychmiast przypisywane i nie są one uwalniane przed ukończeniem zadania. Jaka jest maksymalna liczba zadań, które mogą być jednocześnie realizowane? Jaka jest maksymalna i minimalna liczba napędów, które mogą być pozostawione bezczynne w wyniku takiej polityki?

(b) Zaproponuj odmienną politykę, która poprawiałaby wykorzystanie napędów, a jednocześnie zapobiegała blokowaniu się systemu. Jaka jest maksymalna liczba zadań, które mogą być realizowane jednocześnie? Jakie są granice liczby bezczynnych napędów taśmowych?

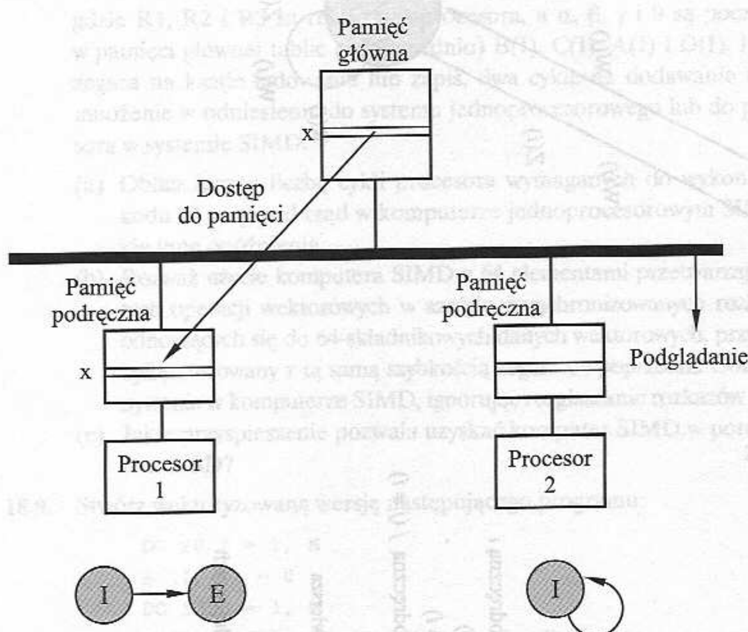
18.3. Czy jesteś w stanie przewidzieć jakąś trudność związaną z zastosowaniem jednoczesnego zapisu w pamięciach podręcznych w przypadku wieloprocesorów opartych na magistrali? Jeśli tak, zaproponuj rozwiązanie.

18.4. Rozważ sytuację, w której 2 procesory skonfigurowane jako SMP co pewien czas wymagają dostępu do tego samego wiersza danych z pamięci głównej. Obydwa procesory

* MIPS oznacza milion rozkazów na sekundę (*przyp. tłum.*).

mają pamięci podręczne i używają protokołu MESI. Początkowo obie pamięci podręczne mają nieważne kopie tego wiersza. Na rysunku 18.20 pokazano konsekwencję odczytu wiersza x przez procesor P1. Jeśli zapoczątkowuje to sekwencję dostępów, nakreśl następne rysunki odpowiadające następującej sekwencji:

1. P2 odczytuje x.
2. P1 zapisuje w x (dla jasności nadaj temu wierszowi w pamięci podręcznej procesora P1 etykietę x').
3. P1 zapisuje w x (nadaj temu wierszowi w pamięci podręcznej procesora P1 etykietę x").
4. P2 odczytuje x.

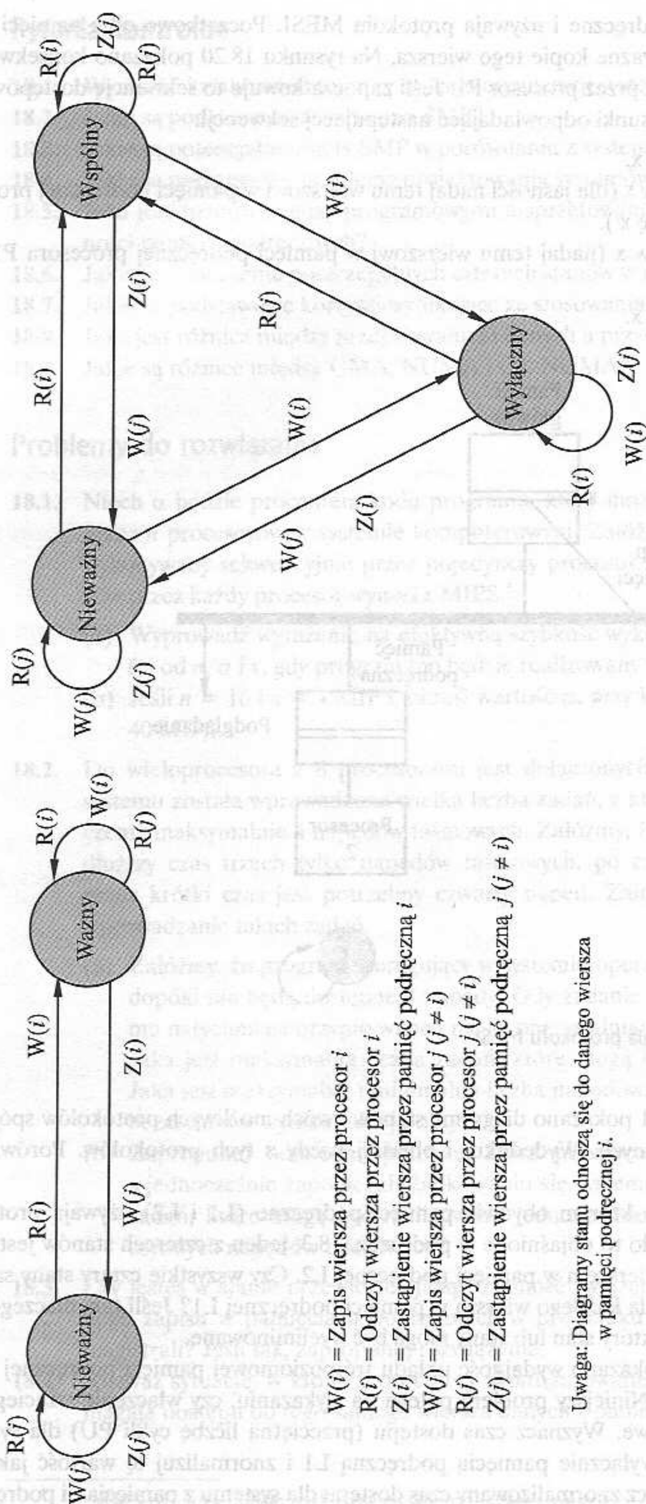


Rysunek 18.20. Przykład działania protokołu MESI

18.5. Na rysunku 18.21 pokazano diagramy stanów dwóch możliwych protokołów spójności pamięci podręcznych. Wydedukuj i objaśnij każdy z tych protokołów. Porównaj je z MESI.

18.6. Rozważ SMP, w którym obydwie pamięci podręczne (L1 i L2) używają protokołu MESI. Jak zostało to objaśnione w podrozdz. 18.3, jeden z czterech stanów jest związany z każdym wierszem w pamięci podręcznej L2. Czy wszystkie cztery stany są również potrzebne dla każdego wiersza w pamięci podręcznej L1? Jeśli tak, dlaczego? Jeśli nie, wyjaśnij, który stan lub stany mogą być wyeliminowane.

18.7. W tabeli 18.1 pokazano wydajność układu trójpoziomowej pamięci podręcznej systemu S/390 IBM. Niniejszy problem polega na wykazaniu, czy włączenie trzeciego poziomu było celowe. Wyznacz czas dostępu (przeciętną liczbę cykli PU) dla systemu dysponującego wyłącznie pamięcią podręczną L1 i znormalizuj tę wartość jako 1,0. Następnie wyznacz znormalizowany czas dostępu dla systemu z pamięciami podręczny-



Rysunek 18.21. Dwa protokoły spójności pamięci podręcznych

mi L1 i L2 oraz dla systemu z wszystkimi trzema pamięciami podręcznymi. Zwróć uwagę na stopień poprawy w każdym przypadku i wyraź swoją opinię na temat wartości pamięci podręcznej L3.

18.8. Poniższy segment kodu musi być wykonany 64 razy w celu obliczenia następującego wyrażenia arytmetyki wektorowej: D(I) = A(I) + B(I) × C(I) dla 0 ≤ I ≤ 63.

```
Load R1,  B(I)            /R1 ← Pamięć (α + I)/
Load R2,  C(I)            /R2 ← Pamięć (β + I)/
Multiply  R1,  R2         /R1 ← (R1) × (R2)/
Load R3,  A(I)            /R3 ← Pamięć (γ + I)/
Add R3,  R1               /R3 ← (R3) + (R1)/
Load D1,  R3              /Pamięć (θ + I) ←(R3)/
```

gdzie R1, R2 i R3 są rejestrami procesora, a α, β, γ i θ są początkowymi adresami w pamięci głównej tablic (odpowiednio) B(I), C(I), A(I) i D(I). Przyjmij cztery cykle zegara na każde ładowanie lub zapis, dwa cykle na dodawanie oraz osiem cykli na mnożenie w odniesieniu do systemu jednoprocesorowego lub do pojedynczego procesora w systemie SIMD.

(a) Oblicz łączną liczbę cykli procesora wymaganych do wykonania tego segmentu kodu 64 razy pod rząd w komputerze jednoprocesorowym SISD, ignorując wszelkie inne opóźnienia.

(b) Rozważ użycie komputera SIMD z 64 elementami przetwarzającymi do wykonania tych operacji wektorowych w sześciu zsynchronizowanych rozkazach wektorowych odnoszących się do 64-składnikowych danych wektorowych, przy czym komputer ten byłby sterowany z tą samą szybkością zegara co poprzedni. Oblicz łączny czas wykonywania w komputerze SIMD, ignorując rozgłaszanie rozkazów i inne opóźnienia.

(c) Jakie przyspieszenie pozwala uzyskać komputer SIMD w porównaniu z komputerem SISD?

18.9. Stwórz wektoryzowaną wersję następującego programu:

```
      DO 20 I = 1, N
      B (I, 1) = 0
      DO 10 J = 1, M
      A(I) = A(I) + B (I, J) × C (I, J)
10    CONTINUE
      D(I) = E(I) + A(I)
20    CONTINUE
```

18.10. Pewien program aplikacyjny jest wykonywany w 9-komputerowym klastrze. Program wzorcowy (*benchmark*) zajął w tym klastrze czas *T*. Ponadto stwierdzono, że 25% *T* to był czas, w którym aplikacja była realizowana jednocześnie we wszystkich dziewięciu komputerach. Przez pozostały czas aplikacja ta musiała być wykonywana na pojedynczym komputerze.

(a) Oblicz efektywne przyspieszenie w porównaniu z wykonywaniem programu na pojedynczym komputerze. Oblicz również α, procent kodu, który był wykonywany równolegle (został zaprogramowany lub skompilowany tak, aby mógł być użyty w trybie klastrowym) w poprzednim programie.

(b) Załóż, że moglibyśmy efektywnie użyć 18 komputerów zamiast 9 w odniesieniu do równolegle wykonywanej części kodu. Oblicz efektywne przyspieszenie w tym przypadku.

18.11. Następujący program FORTRAN ma być zrealizowany w komputerze, jego zaś wersja równoległa – w 32-komputerowym klastrze.

```
L1:        DO 10  I  =  1,  1024
L2:            SUM ( I )  =  0
L3:            DO   20  J  =  1,  I
L4:  20            SUM ( I )  = SUM ( I )  +  I
L5:  10    CONTINUE
```

Załóż, że każdy z wierszy 2 i 4 zajmie dwa cykle maszynowe, łącznie z działaniami procesora i z dostępem do pamięci. Zignoruj nadwyżkę spowodowaną przez instrukcje sterowania pętlą (wiersze 1, 3 i 5) oraz wszystkie inne nadwyżki systemowe i konflikty dotyczące zasobów.

(a) Jaki jest łączny czas wykonywania (w cyklach maszynowych) tego programu w pojedynczym komputerze?

(b) Rozdziel iteracje za pomocą pętli I na 32 komputery w sposób następujący: komputer 1 wykonuje pierwsze 32 iteracje (I = 1 do 32), procesor 2 wykonuje następne 32 iteracje itd. Jakie są czasy wykonywania i współczynnik przyspieszenia w porównaniu z punktem (a)? Weź pod uwagę, że obciążenie obliczeniami dyktowane przez pętlę J nie jest zrównoważone wśród komputerów.

(c) Wyjaśnij, jak można byłoby zmodyfikować równoległe wykonywanie programu, aby uzyskać zrównoważone obciążenie 32 komputerów. Przez obciążenie zrównoważone rozumie się równą liczbę dodawań przypisaną każdemu komputerowi w odniesieniu do obydwu pętli.

(d) Jaki jest minimalny czas wykonywania wynikający z równoległego wykonywania programu w 32 komputerach? Jakie jest efektywne przyspieszenie w stosunku do pojedynczego komputera?

Dodatek A

Logika cyfrowa

A.1. Algebra Boole'a
A.2. Bramki
A.3. Układy kombinacyjne
Implementacja funkcji Boole'a
Multipleksery
Dekodery
Programowalne tablice logiczne (PLA)
Pamięć stała (ROM)
Sumatory
A.4. Układy sekwencyjne
Przerzutniki
Rejestry
Liczniki
A.5. Problemy do rozwiązania

Działanie komputera cyfrowego jest oparte na przechowywaniu i przetwarzaniu danych binarnych. W całej książce zakładaliśmy istnienie elementów pamięciowych występujących w jednym z dwóch stanów stabilnych oraz układów operujących na danych binarnych pod kontrolą sygnałów sterujących w celu realizacji różnych funkcji komputera. W tym dodatku powiemy, w jaki sposób te elementy i układy pamięciowe mogą posłużyć do realizacji cyfrowych układów logicznych, zarówno kombinacyjnych, jak i sekwencyjnych. Rozpoczniemy od krótkiego przeglądu algebry Boole'a, stanowiącej matematyczną podstawę cyfrowych układów logicznych. Następnie wprowadzimy pojęcie bramki. Na zakończenie opiszemy układy kombinacyjne i sekwencyjne zbudowane z bramek.

A.1. Algebra Boole'a

Przy projektowaniu i analizowaniu układów cyfrowych w komputerach i w innych systemach cyfrowych jest używana gałąź matematyki zwana **algebrą Boole'a**. Nazwano ją tak dla uczczenia angielskiego matematyka George'a Boole'a, który zaproponował podstawowe zasady tej algebry w traktacie zatytułowanym *An Investigation of the Laws of Thought on Which to Found the Mathemathical Theories of Logic and Probabilities* (*Badanie praw myśli, które mogą być podstawą matematycznych teorii logiki i prawdopodobieństwa*). W roku 1938 Claude Shannon, asystent na wydziale elektrycznym MIT, zasugerował zastosowanie algebry Boole'a do rozwiązywania problemów projektowania układów przekaźnikowych [SHAN38]. Metody Shannona zostały następnie użyte do analizowania i projektowania elektronicznych układów cyfrowych. Algebra Boole'a jest wygodnym narzędziem w dwóch obszarach:

- **analizy** – jest ekonomicznym sposobem opisywania działania układów cyfrowych;
- **projektowania** – algebra Boole'a może być stosowana do uproszczonej realizacji pożądanych funkcji.

Podobnie jak inne rodzaje algebry, algebra Boole'a używa zmiennych i operacji. W tym przypadku są to logiczne zmienne i operacje. Wobec tego zmienna może przyjmować wartość 1 (PRAWDA) lub 0 (FAŁSZ). Podstawowymi operacjami logicznymi są AND (I), OR (LUB) i NOT (NIE), reprezentowane symbolicznie przez kropkę, znak plus oraz kreskę nad zmienną:

$$\text{A AND B} = \text{A} \cdot \text{B}$$
$$\text{A OR B} = \text{A} + \text{B}$$
$$\text{NOT A} = \overline{\text{A}}$$

Operacja AND daje w wyniku prawdę (wartość binarną 1) wtedy i tylko wtedy, gdy jej obydwa argumenty są prawdziwe. Operacja OR daje w wyniku prawdę, gdy którykolwiek z jej argumentów lub obydwa są prawdziwe. Operacja jednoargumentowa NOT odwraca wartość swojego argumentu. Rozważmy na przykład równanie

$$\text{D} = \text{A} + (\overline{\text{B}} \cdot \text{C})$$

D jest równe 1, jeżeli A jest 1 lub jeśli jednocześnie B=0 i C=1. W przeciwnym razie D jest równe 0.

Oto kilka uwag dotyczących notacji. Przy braku nawiasów operacja AND ma pierwszeństwo przed operacją OR. Gdy nie grozi to dwuznacznością, operację AND zapisuje się przez kolejne wypisywanie argumentów z pominięciem operatora w postaci kropki. Wobec tego

$$A + B \cdot C = A + (B \cdot C) = A + BC$$

Wszystkie te wyrażenia oznaczają: oblicz AND z B i C; następnie oblicz OR z wyniku i z A.

W tabeli A.1 są zdefiniowane podstawowe operacje logiczne w postaci *tablicy prawdy*, w której po prostu wymienia się wartości operacji dla każdej możliwej kombinacji wartości argumentów. W tabeli są również wymienione trzy inne użyteczne operatory: XOR (LUB wykluczające), NAND (NIE-I) oraz NOR (NIE-LUB). Operacja XOR dwóch argumentów logicznych daje w wyniku 1 wtedy i tylko wtedy, gdy dokładnie jeden z argumentów ma wartość 1. Funkcja NAND jest dopełnieniem (NOT) funkcji AND, a NOR jest dopełnieniem OR:

$$A \text{ NAND } B = \text{NOT}(A \text{ AND } B) = \overline{AB}$$
$$A \text{ NOR } B = \text{NOT}(A \text{ OR } B) = \overline{A + B}$$

Jak zobaczymy, te trzy nowe operacje mogą być przydatne przy realizacji pewnych cyfrowych układów logicznych.

Tabela A.1. Operatory boolowskie

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P NAND Q	P NOR Q
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

W tabeli A.2 są podane podstawowe tożsamości algebry Boole'a. Równania ułożono w dwóch kolumnach w celu pokazania komplementarnej (lub dualnej) natury operacji AND i OR. Występują tu dwie klasy tożsamości: reguły podstawowe (zwane *postulatami*), które są podawane bez dowodu, oraz tożsamości pozostałe, które mogą być wyprowadzone na podstawie postulatów podstawowych. Postulaty definiują sposób interpretowania wyrażeń Boole'a. Warto zwrócić uwagę na jedno z dwóch praw rozdzielczości, ponieważ różni się ono od tego, które znajdujemy w zwykłej algebrze:

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Dwa ostatnie wyrażenia w tabeli noszą nazwę prawa De Morgana. Można je również zapisać w następujący sposób:

$$A \text{ NOR } B = \overline{A} \text{ AND } \overline{B}$$
$$A \text{ NAND } B = \overline{A} \text{ OR } \overline{B}$$

Jako ćwiczenie zalecamy zweryfikowanie wyrażeń w tabeli A.2 przez podstawianie rzeczywistych wartości (1 i 0) zamiast zmiennych A, B i C.

Tabela A.2. Podstawowe tożsamości algebry Boole'a

Postulaty podstawowe		
$A \cdot B = B \cdot A$	$A + B = B + A$	Prawa przemienności
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Prawa rozdzielczości
$1 \cdot A = A$	$0 + A = A$	Prawa tożsamości
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Prawa odwrotności
Pozostałe tożsamości		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Prawa łączenia
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	Twierdzenie De Morgana

A.2. Bramki

Podstawowym składnikiem wszystkich cyfrowych układów logicznych jest bramka. Funkcje logiczne są realizowane za pomocą łączenia bramek.

Bramka jest układem elektronicznym, który wytwarza sygnał wyjściowy będący wynikiem operacji Boole'a na sygnałach wejściowych. Podstawowymi bramkami stosowanymi w cyfrowych układach logicznych są: AND, OR, NOT, NAND i NOR. Na rysunku A.1 pokazano te bramki. Każda z nich została zdefiniowana za pomocą trzech sposobów: symbolu graficznego, notacji algebraicznej i tablicy prawdy. Użyta tutaj i w całym dodatku symbolika jest zgodna z normą IEEE Std 91. Zauważmy, że operacja inwersji (NOT) jest oznaczana za pomocą kółka.

Każda z tych bramek ma jedno lub dwa wejścia i jedno wyjście. Gdy są zmieniane wartości na wejściu, prawidłowy sygnał wyjściowy pojawia się niemal natychmiast, po opóźnieniu wynikającym tylko z propagacji sygnałów przez bramkę (zwanym *opóźnieniem bramkowym*). Znaczenie tego omówimy w podrozdz. A.3.

Poza bramkami pokazanymi na rys. A.1, mogą być używane bramki o 3, 4 i o większej liczbie wejść. Dzięki temu wyrażenie X + Y + Z można zrealizować za pomocą jednej bramki OR o 3 wejściach.

Zwykle nie wszystkie bramki są używane do implementacji. Projektowanie i wytwarzanie są prostsze, jeśli używa się tylko jednego lub dwóch rodzajów bramek. Ważne jest więc wyznaczenie *funkcjonalnie pełnych* zbiorów bramek. Oznacza to, że dowolna funkcja Boole'a może być zrealizowana za pomocą bramek pochodzących wyłącznie z takiego zbioru. Następujące zbiory są funkcjonalnie pełne:

- AND, OR, NOT
- AND, NOT
- OR, NOT
- NAND
- NOR

Nazwa	Symbol graficzny	Funkcja algebraiczna	Tablica prawdy
AND	A, B — F	$F = A \cdot B$ lub $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR	A, B — F	$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT	A — F	$F = \overline{A}$ lub $F = A'$	A | F 0 | 1 1 | 0
NAND	A, B — F	$F = \overline{(AB)}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR	A, B — F	$F = \overline{(A + B)}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

Rysunek A.1. Podstawowe bramki logiczne

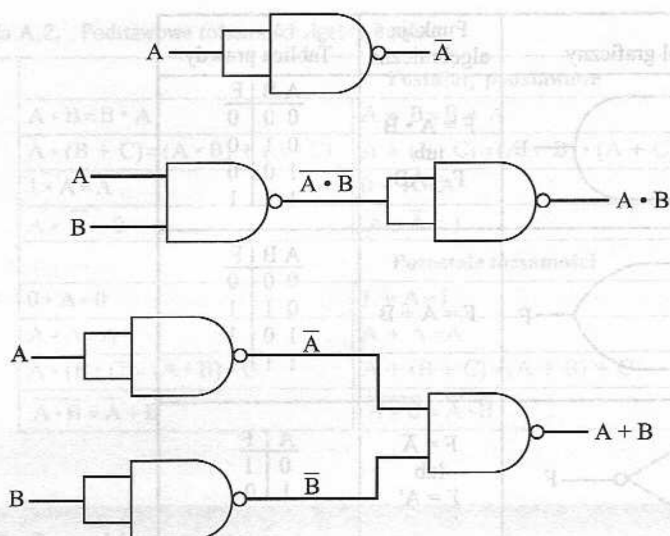
Powinno być oczywiste, że bramki AND, OR i NOT tworzą zbiór funkcjonalnie pełny, ponieważ reprezentują one trzy operacje algebry Boole'a. Aby bramki AND i NOT miały stanowić zbiór funkcjonalnie pełny, musi istnieć sposób syntetyzowania operacji OR za pomocą operacji AND i NOT. Można to uczynić, posługując się prawem De Morgana:

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

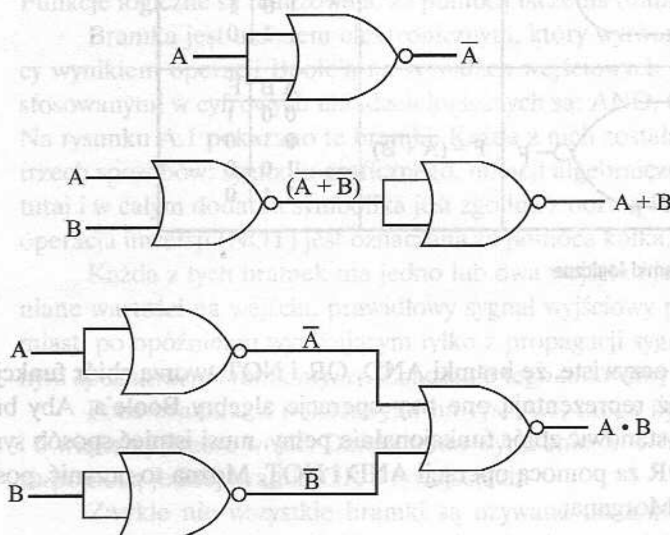
A OR B = NOT((NOT A) AND (NOT B))

Podobnie operacje OR i NOT stanowią zbiór funkcjonalnie pełny, ponieważ mogą one być użyte do syntetyzowania operacji AND.

Na rysunku A.2 widać, w jaki sposób funkcje AND, OR i NOT mogą być zrealizowane wyłącznie za pomocą bramek NAND, a na rys. A.3 jest pokazane to samo za pomocą bramek NOR. Z tego powodu układy cyfrowe mogą być (i często są) realizowane wyłącznie za pomocą bramek NAND lub wyłącznie NOR.



Rysunek A.2. Zastosowanie bramek NAND



Rysunek A.3. Zastosowanie bramek NOR

Zajmując się bramkami, sięgnęliśmy do najbardziej podstawowego poziomu nauki i techniki komputerowej. Analizowanie kombinacji tranzystorów używanych do budowy bramek wykracza poza ten zakres i wchodzi w obszar elektroniki. Dla naszych celów wystarczy jednak opis sposobu używania bramek jako podstawowych elementów służących do budowy podstawowych układów logicznych komputera cyfrowego.

A.3. Układy kombinacyjne

Układ kombinacyjny jest zbiorem wzajemnie połączonych bramek, którego stan wyjść w dowolnej chwili jest wyłącznie funkcją stanu wejść w tej samej chwili. Podobnie jak w przypadku pojedynczej bramki, po ustaleniu stanu na wejściu prawie natychmiast pojawia się sygnał na wyjściu, przy czym występują tylko opóźnienia bramkowe.

Ogólnie rzecz biorąc, układ kombinacyjny zawiera n wejść i m wyjść binarnych. Podobnie jak bramka, układ kombinacyjny może być zdefiniowany przez podanie:

- **Tablicy prawdy.** Dla każdej z 2^n możliwych kombinacji sygnałów wejściowych jest podana wartość binarna każdego z m sygnałów wyjściowych.
- **Symbolu graficznego.** Przedstawiony jest schemat połączeń bramek.
- **Równania Boole'a.** Każdy sygnał wyjściowy jest wyrażony jako funkcja Boole'a sygnałów wejściowych.

Implementacja funkcji Boole'a

Dowolna funkcja Boole'a może być zrealizowana w postaci elektronicznej jako sieć bramek. Dla danej funkcji istnieje pewna liczba rozwiązań alternatywnych. Rozważmy funkcję Boole'a reprezentowaną przez tablicę prawdy w tabeli A.3. Możemy wyrazić tę funkcję po prostu przez wyszczególnienie kombinacji wartości A, B i C, które powodują, że F jest równe 1:

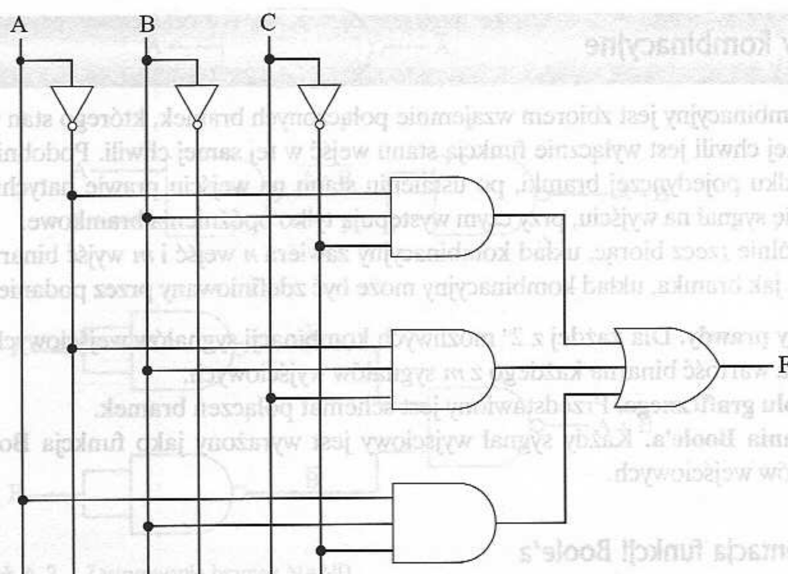
$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} \tag{A.1}$$

Istnieją trzy kombinacje wartości wejściowych, które powodują, że F jest równe 1, i jeśli wystąpi którakolwiek z tych kombinacji, to wynik jest równy 1. Ta forma wyrażenia z oczywistych powodów nazywa się *sumą iloczynów* (*sum of products* – SOP). Na rysunku A.4 jest pokazana bezpośrednia realizacja za pomocą bramek AND, OR i NOT.

Tabela A.3. Funkcja Boole'a trzech zmiennych

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Inny sposób realizacji może być wyprowadzony na podstawie tablicy prawdy. Z postaci wyrażenia (A.1) wynika, że na wyjściu jest 1, jeżeli którakolwiek z kombinacji wejściowych, która daje w wyniku 1, jest prawdziwa. Możemy również powie-



Rysunek A.4. Realizacja tabeli A.3 za pomocą sumy iloczynów

dzieć, że wyjście jest równe 1, jeśli żadna z kombinacji wejściowych dających 0 nie jest prawdziwa. Wobec tego

$$F = (\overline{\overline{A}\,\overline{B}\,\overline{C}}) \cdot (\overline{\overline{A}\,\overline{B}C}) \cdot (\overline{A\overline{B}\,\overline{C}}) \cdot (\overline{A\overline{B}C}) \cdot (\overline{ABC})$$

Można to przekształcić, stosując uogólnienie prawa De Morgana

$$(\overline{X \cdot Y \cdot Z}) = \overline{X} + \overline{Y} + \overline{Z}$$

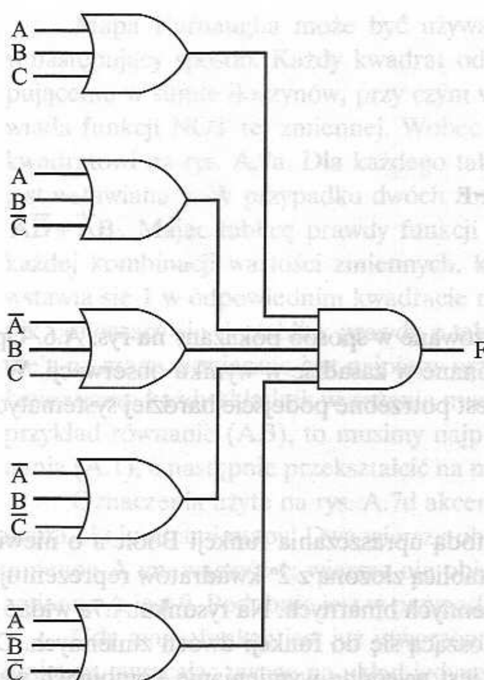
Wobec tego

$$F = (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}}) \cdot (\overline{\overline{A}} + \overline{\overline{B}} + \overline{C}) \cdot (\overline{A} + \overline{\overline{B}} + \overline{\overline{C}}) \cdot (\overline{A} + \overline{\overline{B}} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) =$$
$$= (A + B + C) \cdot (A + B + \overline{C}) \cdot (\overline{A} + B + C) \cdot (\overline{A} + B + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \qquad \text{(A.2)}$$

Jest to postać *iloczynu sum* (*product of sum* – POS), zilustrowana na rys. A.5. Dla jasności nie pokazano bramek NOT. Zamiast tego założono, że osiągalne są: każdy sygnał wejściowy i jego dopełnienie. Upraszcza to schemat logiczny i uwidacznia wejścia bramek.

Funkcja Boole'a może więc być zrealizowana albo w postaci sumy iloczynów, albo iloczynu sum. W tym momencie mogłoby się wydawać, że wybór powinien zależeć od tego, czy tablica prawdy zawiera więcej 1, czy 0 po stronie funkcji wyjściowej. Zastosowanie sumy iloczynów oznacza jedno wyrażenie dla każdej 1, zastosowanie zaś iloczynu sum – jedno wyrażenie dla każdego 0. Należy jednak wziąć pod uwagę, że:

- Na ogół jest możliwe wyprowadzenie prostszego wyrażenia Boole'a na podstawie tablicy prawdy niż na podstawie sumy iloczynów bądź iloczynu sum.



Rysunek A.5. Realizacja tabeli A.3 za pomocą iloczynu sum

- Może być bardziej pożądane zrealizowanie funkcji za pomocą bramek jednego rodzaju (NAND lub NOR).

W odniesieniu do pierwszego punktu jest ważne to, że przy prostszym wyrażeniu Boole'a do realizacji funkcji będzie potrzebnych mniej bramek. W celu uproszczenia mogą być stosowane następujące trzy metody:

- upraszczanie algebraiczne;
- mapy Karnaugha;
- tablice Quine'a-McKluskeya.

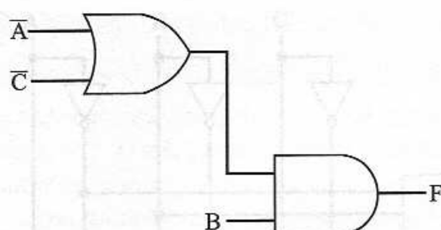
Upraszczanie algebraiczne

Upraszczanie algebraiczne polega na stosowaniu tożsamości przedstawionych w tabeli A.2 do redukowania liczby elementów w wyrażeniach Boole'a. Rozważmy na przykład ponownie równanie (A.1). Równoważnym wyrażeniem jest

$$F = \overline{A}B + B\overline{C} \qquad (A.3)$$

lub nawet prościej

$$F = B(\overline{A} + \overline{C})$$



Rysunek A.6. Uproszczona realizacja tabeli A.3

Wyrażenie to może być zrealizowane w sposób pokazany na rys. A.6. Uproszczenie równania (A.1) zostało wykonane w zasadzie w wyniku obserwacji. W przypadku bardziej złożonych wyrażeń jest potrzebne podejście bardziej systematyczne.

Mapy Karnaugha

Mapa Karnaugha jest wygodną metodą upraszczania funkcji Boole'a o niewielkiej liczbie zmiennych (4÷6). Mapa jest tablicą złożoną z 2^n kwadratów reprezentujących możliwe kombinacje wartości *n* zmiennych binarnych. Na rysunku A.7a widać mapę złożoną z czterech kwadratów, odnoszącą się do funkcji dwóch zmiennych. Do celów wyjaśnionych w dalszym ciągu jest wygodne wymienianie kombinacji wartości zmiennych w porządku 00, 01, 11, 10. Ponieważ kwadraty odpowiadające kombinacjom mają być używane do zapisywania informacji, kombinacje te są zwyczajowo zapisywane nad kwadratami. W przypadku czterech zmiennych mapa składa się z 16 kwadratów ułożonych w sposób pokazany na rys. A.7c.

AB: 00	01	11	10
	1		1

(a) $F = A\overline{B} + \overline{A}B$

A \ BC	00	01	11	10
0			1	1
1				1

(b) $F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$

AB \ CD	00	01	11	10
00			1	
10				
11	1			
10		1		

(c) $F = \overline{A}\overline{B}CD + A\overline{B}\overline{C}D + AB\overline{C}\overline{D}$

		C	C
			B
A			B
A			
	D	D	

(d) Uproszczone etykietowanie mapy

Rysunek A.7. Zastosowanie map Karnaugha do reprezentowania funkcji Boole'a

Mapa Karnaugha może być używana do reprezentowania funkcji Boole'a w następujący sposób. Każdy kwadrat odpowiada unikatowemu iloczynowi występującemu w sumie iloczynów, przy czym wartość 1 odpowiada zmiennej, a 0 odpowiada funkcji NOT tej zmiennej. Wobec tego iloczyn $A\overline{B}$ odpowiada czwartemu kwadratowi na rys. A.7a. Dla każdego takiego iloczynu w odpowiednim kwadracie jest wstawiana 1. W przypadku dwóch zmiennych mapa ta odpowiada więc funkcji $A\overline{B} + \overline{A}B$. Mając tablicę prawdy funkcji Boole'a, łatwo jest zbudować mapę: dla każdej kombinacji wartości zmiennych, która daje w wyniku 1 w tablicy prawdy, wstawia się 1 w odpowiednim kwadracie mapy. Na rysunku A.7b jest pokazany wynik odnoszący się do tablicy prawdy z tabeli A.3. Aby przekształcić wyrażenie Boole'a na mapę, konieczne jest najpierw przetworzenie tego wyrażenia na tzw. postać *kanoniczną*: każdy składnik wyrażenia musi zawierać każdą zmienną. Jeśli mamy na przykład równanie (A.3), to musimy najpierw rozszerzyć je do pełnej postaci równania (A.1), a następnie przekształcić na mapę.

Oznaczenia użyte na rys. A.7d akcentują zależność między zmiennymi a wierszami i kolumnami mapy. Dwa wiersze objęte symbolem A to te wiersze, w których zmienna A ma wartość 1; wiersze nie objęte przez A są tymi wierszami, w których zmienna A jest 0. Podobnie jest w przypadku B, C i D.

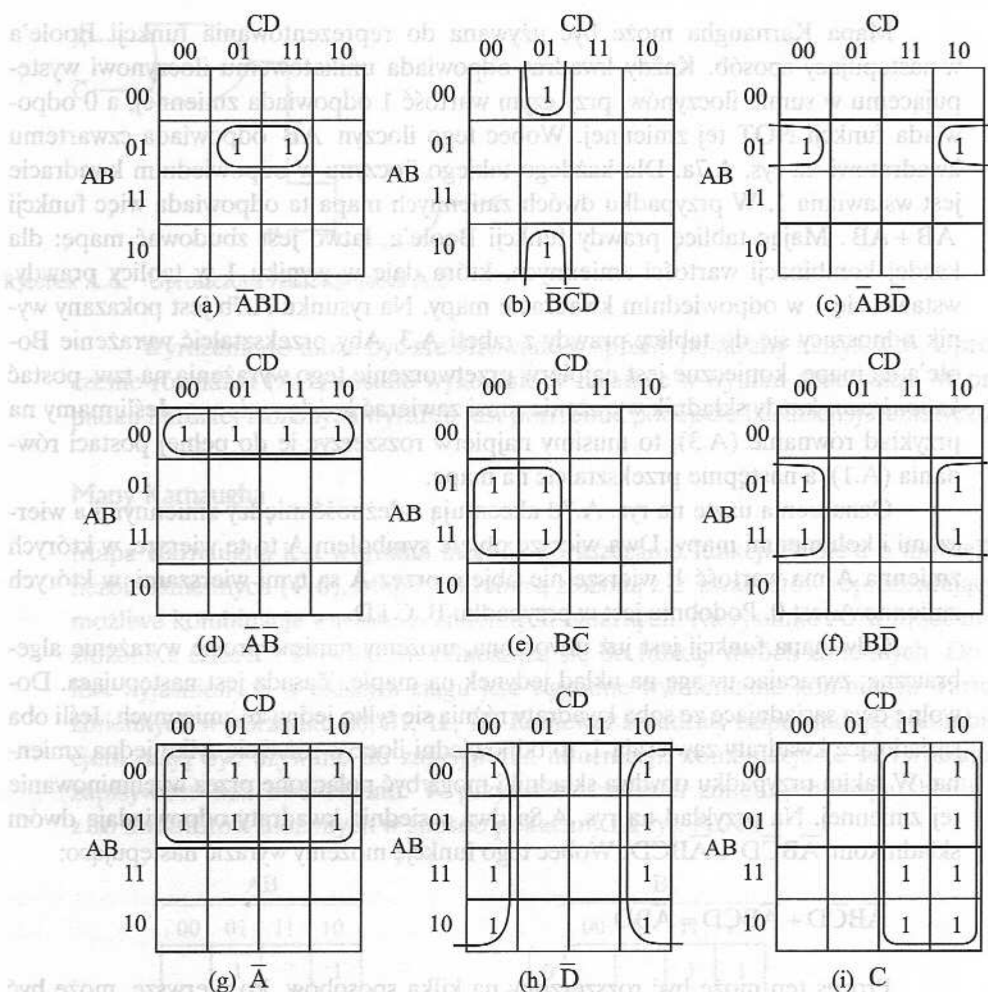
Gdy mapa funkcji jest już utworzona, możemy napisać proste wyrażenie algebraiczne, zwracając uwagę na układ jedynek na mapie. Zasada jest następująca. Dowolne dwa sąsiadujące ze sobą kwadraty różnią się tylko jedną ze zmiennych. Jeśli oba sąsiadujące kwadraty zawierają 1, to odpowiedni iloczyn różni się tylko jedną zmienną. W takim przypadku obydwa składniki mogą być połączone przez wyeliminowanie tej zmiennej. Na przykład na rys. A.8a dwa sąsiednie kwadraty odpowiadają dwóm składnikom $\overline{A}B\overline{C}D$ i $\overline{A}BCD$. Wobec tego funkcję możemy wyrazić następująco:

$$\overline{A}B\overline{C}D + \overline{A}BCD = \overline{A}BD$$

Proces ten może być rozszerzony na kilka sposobów. Po pierwsze, może być rozszerzona koncepcja przyległości przez uwzględnienie sąsiedztwa wokół krawędzi mapy. Wobec tego górny kwadrat kolumny jest sąsiedni w stosunku do dolnego kwadratu, a lewy kwadrat w wierszu sąsiaduje z prawym. Warunki te są zilustrowane na rys. A.8b i c. Po drugie, możemy zgrupować nie tylko 2 kwadraty, lecz 2^n sąsiednich kwadratów, tzn. 4, 8 itd. Następne trzy przykłady na rys. A.8 pokazują grupowanie 4 kwadratów. Zauważmy, że w tym przypadku dwie spośród zmiennych mogą być wyeliminowane. W ostatnich trzech przykładach pokazano grupowanie 8 kwadratów, co umożliwia wyeliminowanie trzech zmiennych.

Reguły upraszczania możemy podsumować następująco:

1. Wśród zaznaczonych kwadratów (kwadraty z 1) znajdź te, które należą do unikatowego, największego bloku składającego się z 1, 2, 4 lub 8 kwadratów i zakreśl te bloki.
2. Wybierz dodatkowe bloki zaznaczonych kwadratów, które są możliwie duże i możliwie nieliczne, jednak uwzględniaj każdy zaznaczony kwadrat przynajmniej raz.

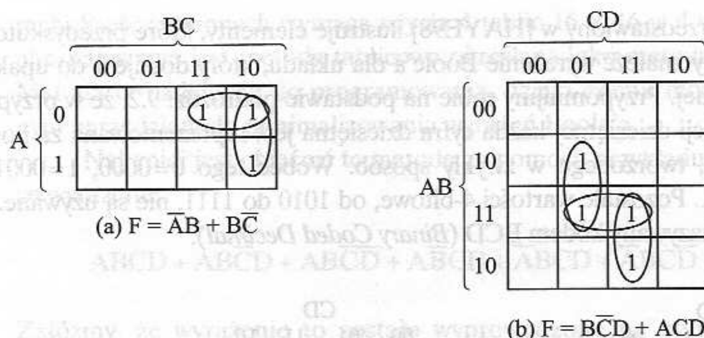


Rysunek A.8. Zastosowanie map Karnaugha

W niektórych przypadkach wyniki mogą nie być unikatowe. Jeśli na przykład zaznaczony kwadrat łączy się z dokładnie dwoma innymi kwadratami, i nie ma czwartego kwadratu umożliwiającego skompletowanie większej grupy, trzeba dokonać wyboru jednego z dwóch ugrupowań. Zakreślając grupy, masz możliwość użycia tej samej wartości 1 więcej niż raz.

3. Kontynuuj kreślenie pętli wokół pojedynczych zaznaczonych kwadratów lub par sąsiadujących ze sobą zaznaczonych kwadratów, lub grup po 4, 8 itd., w taki sposób, że każdy zaznaczony kwadrat będzie należał przynajmniej do jednej pętli; następnie użyj możliwie małej liczby tych bloków, aby objąć wszystkie zaznaczone kwadraty.

Proces ten został zilustrowany na rys. A.9a opartym na tabeli A.3. Jeśli którakolwiek spośród izolowanych jedynek pozostaje poza ugrupowaniem, to każda z nich jest zakreślana jako grupa. Na zakończenie, zanim przejdziemy od mapy do



Rysunek A.9. Pokrywające się grupy

uproszczonego wyrażenia Boole'a, możemy wyeliminować dowolną z grup jedynek, która w całości pokrywa się z innymi grupami (patrz rys. A.9b). W tym przypadku pozioma grupa pokrywa się z innymi i przy tworzeniu wyrażenia Boole'a może być zignorowana.

Warto zwrócić uwagę na pewną dodatkową właściwość map Karnaugha. W niektórych przypadkach pewne kombinacje wartości zmiennych nigdy nie występują i wobec tego nie występują odpowiadające im sygnały wyjściowe. Te warunki określa się jako „bez znaczenia" (*don't care*). W przypadku każdego z takich warunków do odpowiedniego kwadratu mapy wprowadza się literę „d". Podczas grupowania i upraszczania każde „d" może być traktowane jako 1 lub 0, zależnie od tego, co prowadzi do prostszego wyrażenia.

Tabela A.4. Tablica prawdy jednocyfrowego rejestru następnikowego w notacji upakowanej dziesiętnej

Liczba	Wejście				Liczba	Wyjście			
	A	B	C	D		W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
Warunki „bez znaczenia"	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	d

Przykład przedstawiony w [HAYE98] ilustruje elementy, które przedyskutowaliśmy. Chcielibyśmy znaleźć wyrażenie Boole'a dla układu, który dodaje 1 do upakowanej cyfry dziesiętnej. Przypomnijmy sobie na podstawie podrozdz. 9.2, że w przypadku upakowanej notacji dziesiętnej każda cyfra dziesiętna jest reprezentowana za pomocą kodu 4-bitowego, tworzonego w zwykły sposób. Wobec tego 0=0000, 1=0001, ..., 8=1000 i 9=1001. Pozostałe wartości 4-bitowe, od 1010 do 1111, nie są używane. Kod ten jest również nazywany kodem BCD (*Binary Coded Decimal*).

AB \ CD	00	01	11	10
00				
01			1	
11	d	d	d	d
10	1		d	d

(a) $W = A\overline{D} + \overline{A}BCD$

AB \ CD	00	01	11	10
00			1	
01	1	1		1
11	d	d	d	d
10			d	d

(b) $X = B\overline{D} + B\overline{C} + \overline{B}CD$

AB \ CD	00	01	11	10
00		1		1
01		1		1
11	d	d	d	d
10			d	d

(c) $Y = \overline{A}\overline{C}D + \overline{A}C\overline{D}$

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	d	d	d	d
10	1		d	d

(d) $Z = \overline{D}$

Rysunek A.10. Mapy Karnaugha dla układu inkrementującego

W tabeli A.4 znajduje się tablica prawdy odnosząca się do 4-bitowego wyniku, który jest o 1 większy niż 4-bitowe dane wejściowe BCD. Dodawanie to jest wykonywane modulo 10, więc 9 + 1 = 0. Zauważmy również, że sześć spośród kodów wejściowych daje wyniki „bez znaczenia", ponieważ nie są to ważne dane wejściowe BCD. Na rysunku A.10 jest pokazana wynikowa mapa Karnaugha dla każdej ze zmiennych wyjściowych. Kwadraty „d" zostały użyte w celu osiągnięcia możliwie najlepszych ugrupowań.

Metoda Quine'a-McKluskeya

W przypadku więcej niż 4 zmiennych mapa Karnaugha staje się coraz bardziej kłopotliwa. Dla 5 zmiennych są potrzebne dwie mapy 16 × 16, przy czym przy analizowaniu przyległości jedna z map jest rozpatrywana jako leżąca nad drugą w 3 wymia-

rach. Sześć zmiennych wymaga użycia 4 tablic 16 × 16 w 4 wymiarach! Podejściem alternatywnym jest metoda tablicowa określana jako metoda Quine'a-McKluskeya. Metoda ta nadaje się do programowania, dzięki czemu może być zautomatyzowanym narzędziem do minimalizowania wyrażeń Boole'a.

Najlepiej jest objaśnić tę metodę za pomocą przykładu. Rozważmy następujące wyrażenie:

$$ABCD + AB\overline{C}D + AB\overline{C}\,\overline{D} + A\overline{B}CD + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D$$

Załóżmy, że wyrażenie to zostało wyprowadzone na podstawie tablicy prawdy. Chcielibyśmy utworzyć wyrażenie minimalne odpowiednie do realizacji za pomocą bramek.

Pierwszym krokiem jest zbudowanie tablicy, w której każdy wiersz odpowiada jednemu z członów wyrażenia. Człony są grupowane zależnie od liczby zanegowanych zmiennych. Rozpoczynamy od członu nie zawierającego zmiennych zanegowanych, jeśli taki istnieje, następnie bierzemy pod uwagę wszystkie iloczyny z jedną zmienną zanegowaną i tak dalej. W tabeli A.5 jest pokazana lista dotycząca naszego przykładu, przy czym wiersze tabeli pokazują grupowanie. Dla jasności, każdy człon jest reprezentowany przez 1 odpowiadającą każdej zmiennej niezanegowanej oraz przez 0 dla każdej zanegowanej. Grupujemy więc człony zależnie od liczby jedynek, jaką zawierają. Wskaźnik kolumny jest po prostu równoważnikiem dziesiętnym i będzie użyteczny w dalszym ciągu.

Tabela A.5. Pierwszy etap metody Quine'a-McKluskeya dla wyrażenia
$F = ABCD + AB\overline{C}D + AB\overline{C}\,\overline{D} + A\overline{B}CD + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D$

Iloczyn	Indeks	A	B	C	D	
$\overline{A}\,\overline{B}\,\overline{C}D$	1	0	0	0	1	√
$\overline{A}B\overline{C}D$	5	0	1	0	1	√
$\overline{A}BC\overline{D}$	6	0	1	1	0	√
$AB\overline{C}\,\overline{D}$	12	1	1	0	0	√
$\overline{A}BCD$	7	0	1	1	1	√
$A\overline{B}CD$	11	1	0	1	1	√
$AB\overline{C}D$	13	1	1	0	1	√
$ABCD$	15	1	1	1	1	√

Następnym krokiem jest znalezienie wszystkich par iloczynów, które różnią się tylko jedną zmienną; to znaczy wszystkich par iloczynów, które są takie same z wyjątkiem tego, że jedna ze zmiennych jest 0 w jednym z iloczynów, a 1 w pozostałych. Ze względu na sposób, w jaki pogrupowaliśmy iloczyny, możemy to uczynić, rozpoczynając od pierwszej grupy i porównując każdy iloczyn z pierwszej grupy z każdym iloczynem z drugiej grupy. Następnie porównujemy każdy iloczyn z drugiej grupy ze wszystkimi iloczynami trzeciej grupy i tak dalej. Gdy tylko stwierdza się zgodność, umieszcza się znak (√) za każdym członem, łączy się parę (eliminując

zmienną, która różni się w obydwu członach), a wynik dodaje się do nowej listy. Na przykład człony $\overline{A}BC\overline{D}$ i $\overline{A}BCD$ są łączone, co daje ABC. Kontynuuje się ten proces, aż do przebadania całej oryginalnej tablicy. Wynikiem jest nowa tablica o następujących zapisach:

$\overline{A}\overline{C}D$	$AB\overline{C}$	ABD √
	$B\overline{C}D$ √	ACD
	$\overline{A}BC$	BCD √
	$\overline{A}BD$ √	

Nowa tablica jest podzielona na grupy w ten sam sposób jak pierwsza. Następnie jest ona przetwarzana w ten sam sposób jak pierwsza. Oznacza to, że są zaznaczane iloczyny różniące się tylko jedną zmienną i w trzeciej tablicy jest tworzony nowy iloczyn. W naszym przykładzie trzecia tablica zawiera tylko jeden iloczyn:

BD

Opisany proces jest kontynuowany poprzez kolejne tablice, aż zostanie utworzona tablica nie wykazująca żadnych zgodności. W naszym przypadku proces ten objął trzy tablice.

Tabela A.6. Ostatni etap metody Quine'a-McKluskeya dla wyrażenia
$F = ABCD + AB\overline{C}D + AB\overline{C}\overline{D} + A\overline{B}CD + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}\overline{B}\overline{C}D$

	$ABCD$	$AB\overline{C}D$	$AB\overline{C}\overline{D}$	$A\overline{B}CD$	$\overline{A}BCD$	$\overline{A}BC\overline{D}$	$\overline{A}B\overline{C}D$	$\overline{A}\overline{B}\overline{C}D$
BD	×	×			×		×	
$\overline{A}\overline{C}D$							⊠	⊗
$\overline{A}BC$					⊠	⊗		
$AB\overline{C}$		⊠	⊗					
ACD	⊠			⊗				

W wyniku tego procesu wyeliminowaliśmy wiele z możliwych członów wyrażenia. Te człony, które nie zostały wyeliminowane, są używane do zbudowania tablicy, takiej jak pokazana w tabeli A.6. Każdy wiersz tej tablicy odpowiada jednemu członowi, który nie został wyeliminowany (nie został zaznaczony) w którejkolwiek z dotychczasowych tablic. Każda kolumna odpowiada jednemu członowi w wyrażeniu oryginalnym. Na przecięciu wiersza i kolumny stawia się znak × wskazujący „zgodność". Oznacza on, że zmienne występujące w elemencie wiersza mają te same wartości co zmienne występujące w elemencie kolumny. Następnie zakreślamy kółkiem każdy znak ×, który występuje pojedynczo w kolumnie. Każdy × w wierszu, w którym występuje × zakreślony kółkiem, zaznaczamy za pomocą kwadratu. Gdy każda kolumna ma teraz × zaznaczone kwadratem lub kółkiem, to znaczy, że zada-

nie zostało zakończone. Elementy odpowiadające wierszom, których znaki × zostały zaznaczone, składają się na wyrażenie minimalne. W naszym przykładzie ostatecznym wyrażeniem jest więc

$$AB\overline{C} + ACD + \overline{A}BC + \overline{A}\overline{C}D$$

W przypadkach gdy niektóre kolumny nie zawierają kółka ani kwadratu, jest wymagane dodatkowe przetwarzanie. W istocie kontynuujemy dodawanie wierszy, aż we wszystkich kolumnach znajdzie się znak ×.

Podsumujmy teraz metodę Quine'a-McKluskeya, próbując intuicyjnie uzasadnić jej funkcjonowanie. Pierwsza faza operacji wydaje się prosta. Proces polega na eliminowaniu niepotrzebnych zmiennych w poszczególnych członach. Wobec tego wyrażenie $ABC + AB\overline{C}$ jest równoważne AB, ponieważ

$$ABC + AB\overline{C} = AB(C + \overline{C}) = AB$$

Po wyeliminowaniu zmiennych pozostało nam wyrażenie, które w oczywisty sposób jest równoważne wyrażeniu oryginalnemu. Mogą w nim jednak występować człony nadmiarowe, podobnie jak nadmiarowe grupy w mapach Karnaugha. Ostatnia tablica pozwala na uwzględnienie każdego iloczynu wyrażenia oryginalnego w ten sposób, że liczba iloczynów wyrażenia finalnego jest minimalna.

Implementacja NAND i NOR

Innym problemem implementacji funkcji Boole'a jest rodzaj użytych bramek. Często jest pożądana realizacja funkcji Boole'a wyłącznie za pomocą bramek NAND lub wyłącznie NOR. Chociaż może to nie być realizacja o minimalnej liczbie bramek, ma ona zaletę regularności, co może uprościć proces wytwarzania. Rozważmy ponownie równanie (A.3):

$$F = B(\overline{A} + \overline{C})$$

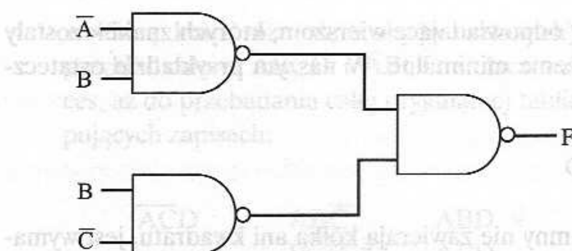
Ponieważ dopełnienie dopełnienia jest po prostu wartością początkową, można napisać

$$F = B(\overline{A} + \overline{C}) = \overline{\overline{(\overline{A}B) + (B\overline{C})}}$$

Po zastosowaniu prawa De Morgana otrzymujemy

$$F = \overline{\overline{(\overline{A}B)} \cdot \overline{(B\overline{C})}}$$

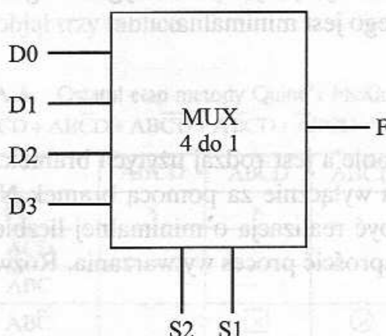
Do realizacji tego wyrażenia są więc potrzebne trzy bramki NAND, co widać na rys. A.11.



Rysunek A.11. Implementacja tabeli A.3 za pomocą bramek NAND

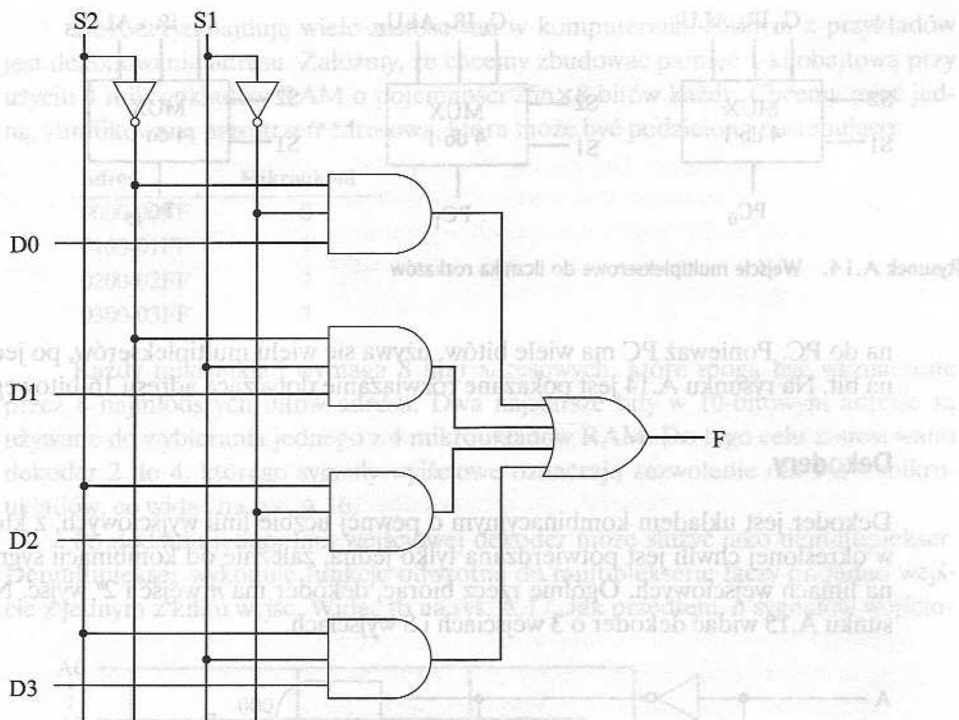
Multipleksery

Multiplekser łączy wiele wejść z jednym wyjściem. W dowolnej chwili jedno z tych wejść jest wybrane jako połączone z wyjściem. Ogólny schemat blokowy jest pokazany na rys. A.12. Przedstawia on multiplekser 4 do 1 o czterech liniach wejściowych, oznaczonych D0, D1, D2 i D3. Na wyjście F multipleksera jest dostarczany sygnał z jednej wybranej linii wejściowej. Aby wybrać jedno z 4 możliwych wejść, jest potrzebny 2-bitowy kod wyboru realizowany za pomocą 2 linii wyboru oznaczonych S1 i S2.



Rysunek A.12. Reprezentacja multipleksera 4 do 1

Przykład multipleksera 4 do 1 został zdefiniowany za pomocą tablicy prawdy w tabeli A.7. Jest to uproszczona postać tablicy prawdy. Zamiast pokazywać wszystkie możliwe kombinacje zmiennych wejściowych, zawiera ona sygnały wyjściowe jako dane z linii wejściowej D0, D1, D2 lub D3. Na rysunku A.13 widać realizacje przy użyciu bramek AND, OR i NOT. Linie S1 i S2 są połączone z bramkami AND w taki sposób, że dla dowolnej kombinacji S1 i S2 trzy z bramek AND dadzą sygnał wyjściowy 0. Na wyjściu czwartej bramki AND pojawi się sygnał wybranej linii, który może być albo 0, albo 1. Wobec tego na trzech wejściach bramki OR są zawsze 0, a stan wyjścia bramki OR będzie zawsze równy wartości wybranej bramki wejściowej. Przy zastosowaniu tej regularnej organizacji łatwo jest zbudować multipleksery o rozmiarach 8 do 1, 16 do 1 i tak dalej.



Rysunek A.13. Realizacja multipleksera

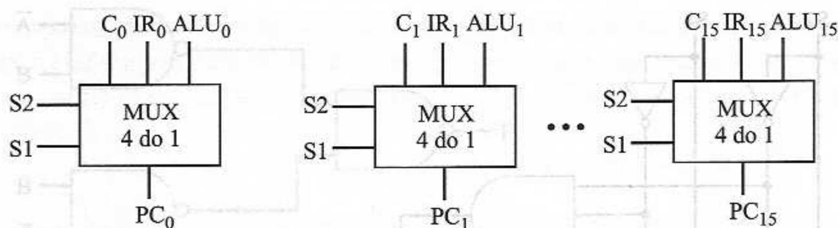
Tabela A.7. Tablica prawdy multipleksera 4 do 1

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Multipleksery są używane w układach cyfrowych do sterowania przepływem sygnałów i danych. Przykładem jest ładowanie licznika programu (PC). Wartość, która ma być załadowana do licznika programu, może pochodzić z jednego z wielu różnych źródeł:

- licznika binarnego, jeśli stan PC ma być inkrementowany w celu określenia kolejnego rozkazu;
- rejestru rozkazu, jeśli został właśnie wykonany rozkaz rozgałęzienia używający adresu bezpośredniego;
- wyjścia ALU, jeśli rozkaz rozgałęzienia określa adres w trybie indeksowym.

Źródła te mogą być połączone z liniami wejściowymi multipleksera, przy czym PC jest połączony z linią wyjściową. Linie wyboru określają, która wartość jest ładowa-

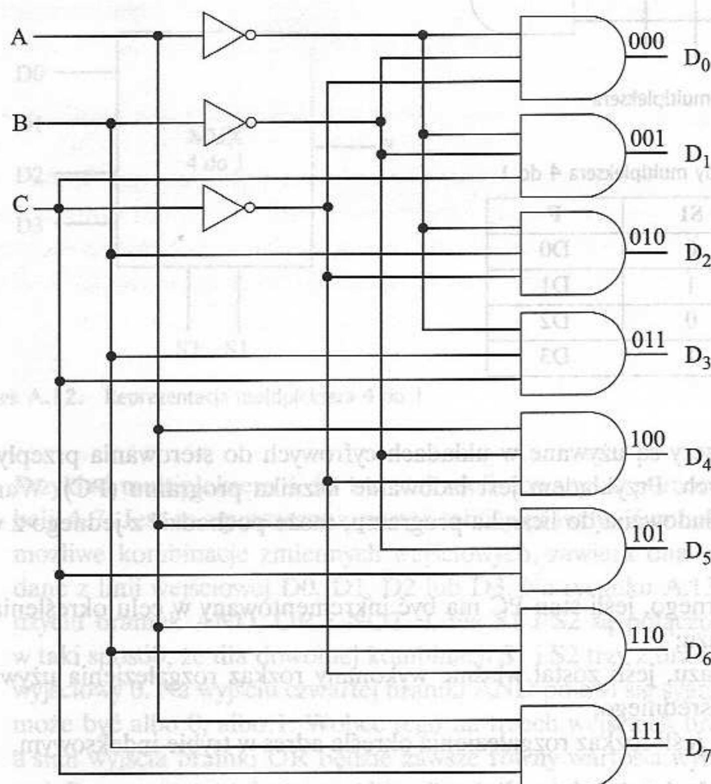


Rysunek A.14. Wejście multiplekserowe do licznika rozkazów

na do PC. Ponieważ PC ma wiele bitów, używa się wielu multiplekserów, po jednym na bit. Na rysunku A.14 jest pokazane rozwiązanie dotyczące adresu 16-bitowego.

Dekodery

Dekoder jest układem kombinacyjnym o pewnej liczbie linii wyjściowych, z których w określonej chwili jest potwierdzana tylko jedna, zależnie od kombinacji sygnałów na liniach wejściowych. Ogólnie rzecz biorąc, dekoder ma n wejść i 2^n wyjść. Na rysunku A.15 widać dekoder o 3 wejściach i 8 wyjściach.



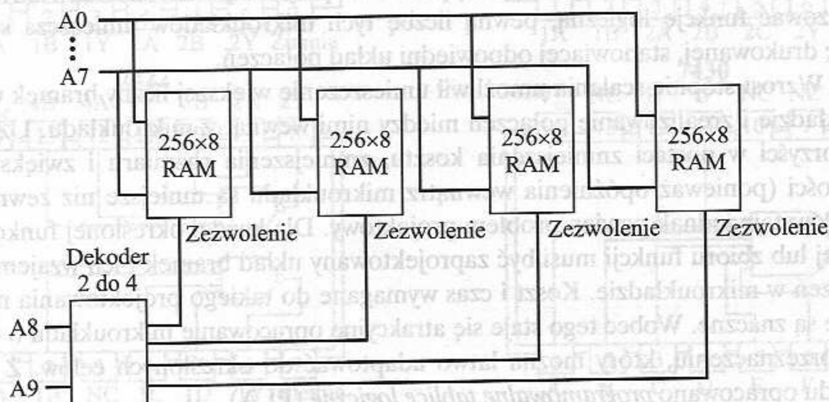
Rysunek A.15. Dekoder z trzema wejściami i $2^3 = 8$ wyjściami

Dekodery znajdują wiele zastosowań w komputerach. Jednym z przykładów jest dekodowanie adresu. Załóżmy, że chcemy zbudować pamięć 1-kilobajtową przy użyciu 4 mikroukładów RAM o pojemności 256×8 bitów każdy. Chcemy mieć jedną, zunifikowaną przestrzeń adresową, która może być podzielona następująco:

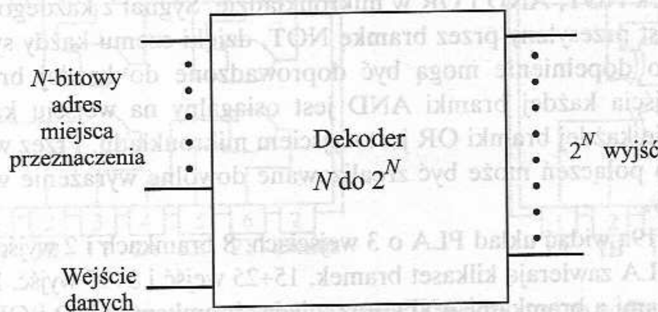
Adres	Mikroukład
0000-00FF	0
0100-01FF	1
0200-02FF	2
0300-03FF	3

Każdy mikroukład wymaga 8 linii adresowych, które mogą być wyznaczone przez 8 najmłodszych bitów adresu. Dwa najstarsze bity w 10-bitowym adresie są używane do wybierania jednego z 4 mikroukładów RAM. Do tego celu zastosowano dekoder 2 do 4, którego sygnały wyjściowe oznaczają zezwolenie dla 1 z 4 mikroukładów, co widać na rys. A.16.

Po dodaniu jednej linii wejściowej dekoder może służyć jako demultiplekser. Demultiplekser wykonuje funkcję odwrotną do multipleksera; łączy on jedno wejście z jednym z kilku wyjść. Widać to na rys. A.17. Jak przedtem, n sygnałów wejścio-



Rysunek A.16. Dekodowanie adresu



Rysunek A.17. Realizacja demultipleksera za pomocą dekodera

ciowych po zdekodowaniu daje jeden z 2^n sygnałów wyjściowych. Wszystkie 2^n linii wyjściowych połączono z linią wejściową danych w bramce AND. Wobec tego n wejść działa jako adres służący do wybrania określonej linii wyjściowej, a wartość wejściowej linii danych (0 lub 1) jest kierowana do tej właśnie linii wyjściowej.

Konfiguracja pokazana na rys. A.17 może być postrzegana w inny sposób. Zmieńmy nazwę nowej linii z *wejścia danych* na *zezwolenie*. Umożliwia to taktowanie dekodera. Zdekodowane sygnały wyjściowe pojawiają się tylko wtedy, kiedy są doprowadzone zakodowane sygnały wejściowe i jednocześnie na linii zezwolenia jest wartość 1.

Programowalne tablice logiczne (PLA)

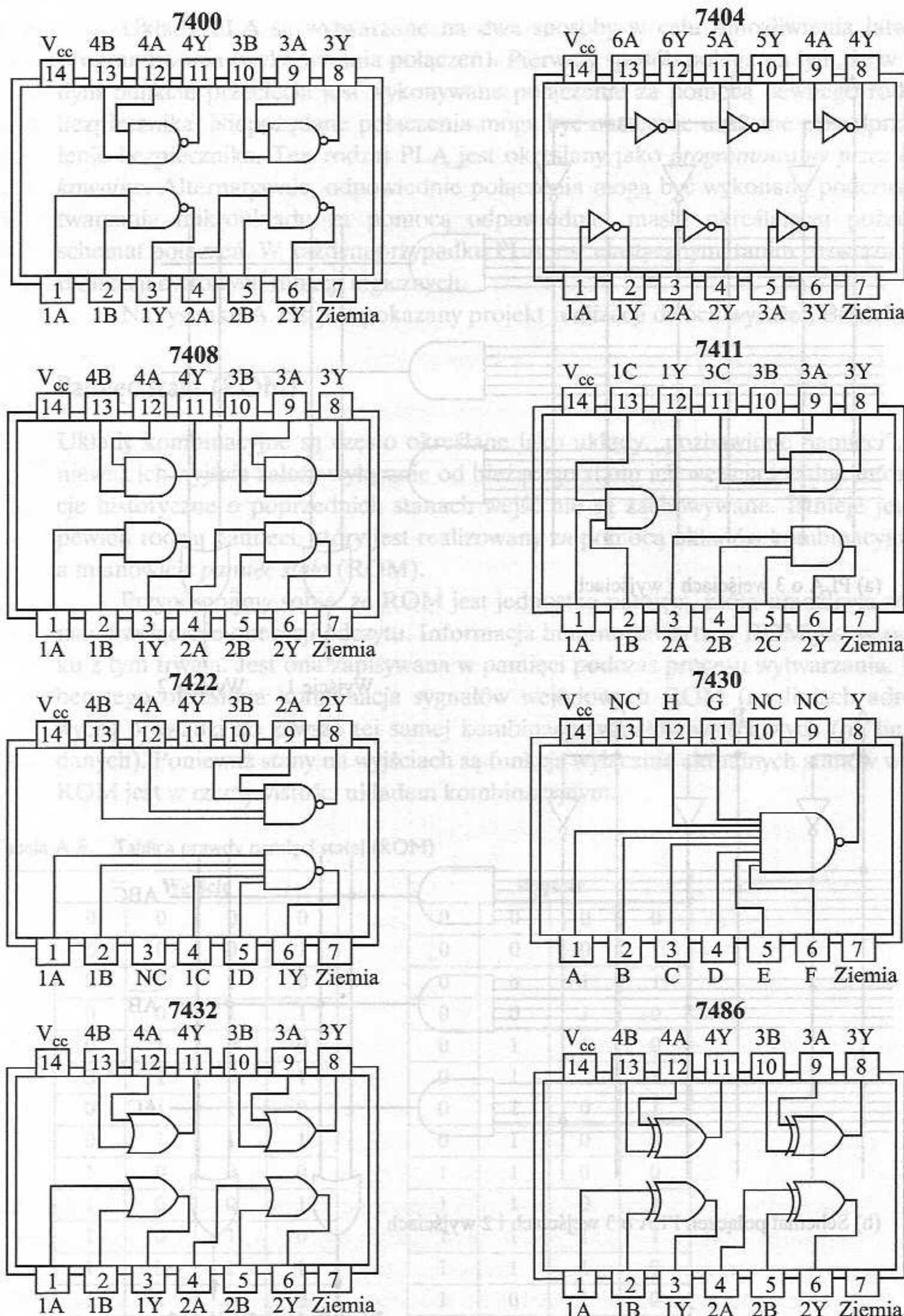
Dotychczas traktowaliśmy pojedyncze bramki jako elementy, za pomocą których mogą być realizowane dowolne funkcje. Projektant dążył do zminimalizowania liczby bramek przez przekształcanie odpowiednich wyrażeń Boole'a.

W miarę wzrostu stopnia scalenia układów pojawia się możliwość innego podejścia. Wczesne układy scalone o małym stopniu scalenia (SSI) zawierały 1÷10 bramek w mikroukładzie. Każda bramka jest traktowana niezależnie, w sposób dotychczas opisany. Na rysunku A.18 są pokazane przykłady mikroukładów SSI. Aby zrealizować funkcję logiczną, pewną liczbę tych mikroukładów umieszcza się na płytce drukowanej, stanowiącej odpowiedni układ połączeń.

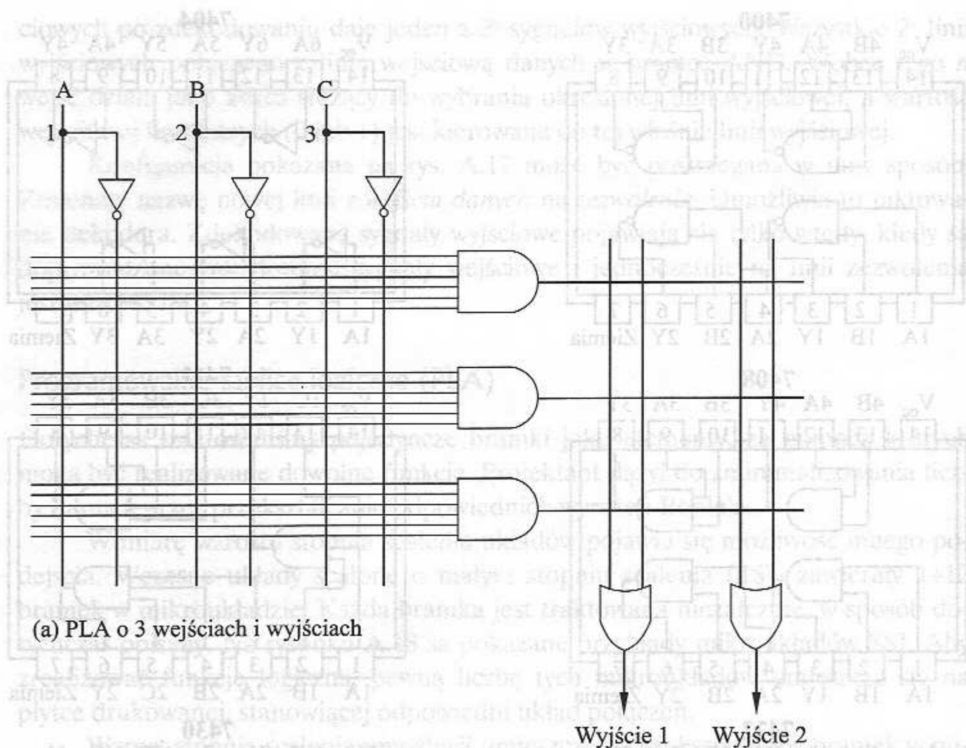
Wzrost stopnia scalenia umożliwił umieszczenie większej liczby bramek w mikroukładzie i zrealizowanie połączeń między nimi wewnątrz mikroukładu. Uzyskano korzyści w postaci zmniejszenia kosztu, zmniejszenia rozmiaru i zwiększenia szybkości (ponieważ opóźnienia wewnątrz mikroukładu są mniejsze niż zewnętrzne). Powstaje jednak pewien problem projektowy. Dla każdej określonej funkcji logicznej lub zbioru funkcji musi być zaprojektowany układ bramek i ich wzajemnych połączeń w mikroukładzie. Koszt i czas wymagane do takiego projektowania na życzenie są znaczne. Wobec tego staje się atrakcyjne opracowanie mikroukładu o ogólnym przeznaczeniu, który można łatwo adaptować do określonych celów. Z tego powodu opracowano *programowalne tablice logiczne* (PLA).

Koncepcja PLA jest oparta na tym, że dowolna funkcja Boole'a (tablica prawdy) może być wyrażona w postaci sumy iloczynów. Tablica PLA jest regularnym układem bramek NOT, AND i OR w mikroukładzie. Sygnał z każdego wejścia mikroukładu jest przesyłany przez bramkę NOT, dzięki czemu każdy sygnał wejściowy oraz jego dopełnienie mogą być doprowadzone do każdej bramki AND. Sygnał z wyjścia każdej bramki AND jest osiągalny na wejściu każdej bramki OR, a wyjście każdej bramki OR jest wyjściem mikroukładu. Przez wykonanie odpowiednich połączeń może być zrealizowane dowolne wyrażenie w postaci sumy iloczynów.

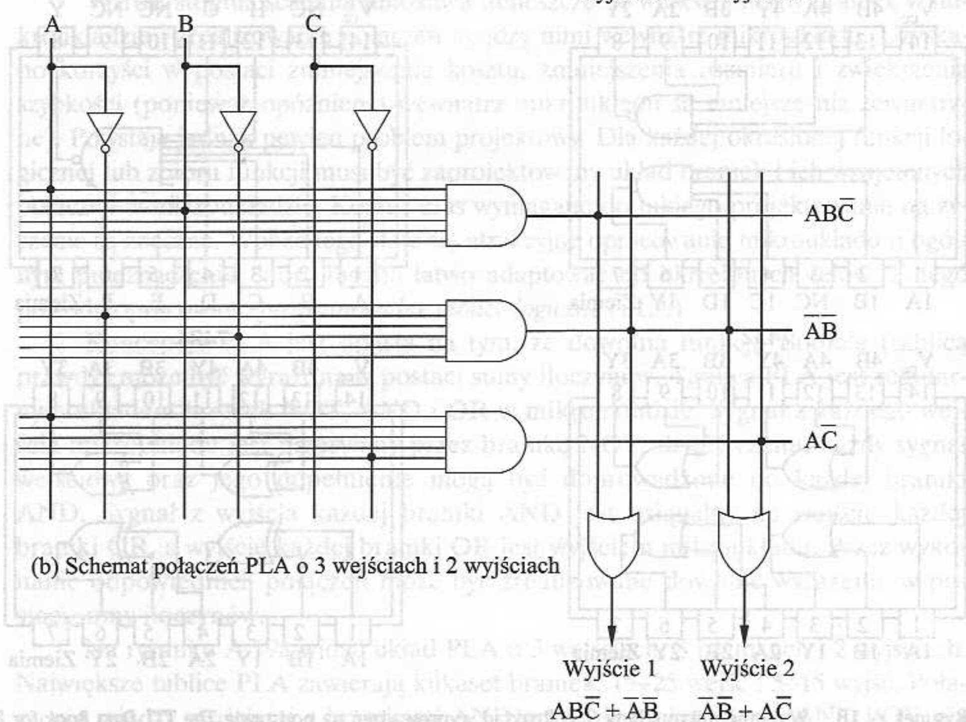
Na rysunku A.19a widać układ PLA o 3 wejściach, 8 bramkach i 2 wyjściach. Największe tablice PLA zawierają kilkaset bramek, 15÷25 wejść i 5÷15 wyjść. Połączenia między wejściami a bramkami AND oraz między bramkami AND i OR nie zostały pokazane.



Rysunek A.18. Wybrane mikroukłady SSI. Rozkład wyprowadzeń na podstawie *The TTL Data Book for Design Engineers*, ©1976 Texas Instruments Incorporated



(a) PLA o 3 wejściach i wyjściach



(b) Schemat połączeń PLA o 3 wejściach i 2 wyjściach

Rysunek A.19. Przykład programowalnej tablicy logicznej

Układy PLA są wytwarzane na dwa sposoby w celu umożliwienia łatwego programowania (wykonywania połączeń). Pierwszy sposób polega na tym, że w każdym punkcie przecięcia jest wykonywane połączenie za pomocą pewnego rodzaju bezpiecznika. Niepożądane połączenia mogą być następnie usuwane przez przepalenie bezpiecznika. Ten rodzaj PLA jest określany jako *programowalny przez użytkownika*. Alternatywnie, odpowiednie połączenia mogą być wykonane podczas wytwarzania mikroukładu za pomocą odpowiedniej maski określającej pożądany schemat połączeń. W każdym przypadku PLA jest elastycznym, tanim rozwiązaniem realizacji cyfrowych funkcji logicznych.

Na rysunku A.19b jest pokazany projekt realizacji dwóch wyrażeń Boole'a.

Pamięć stała (ROM)

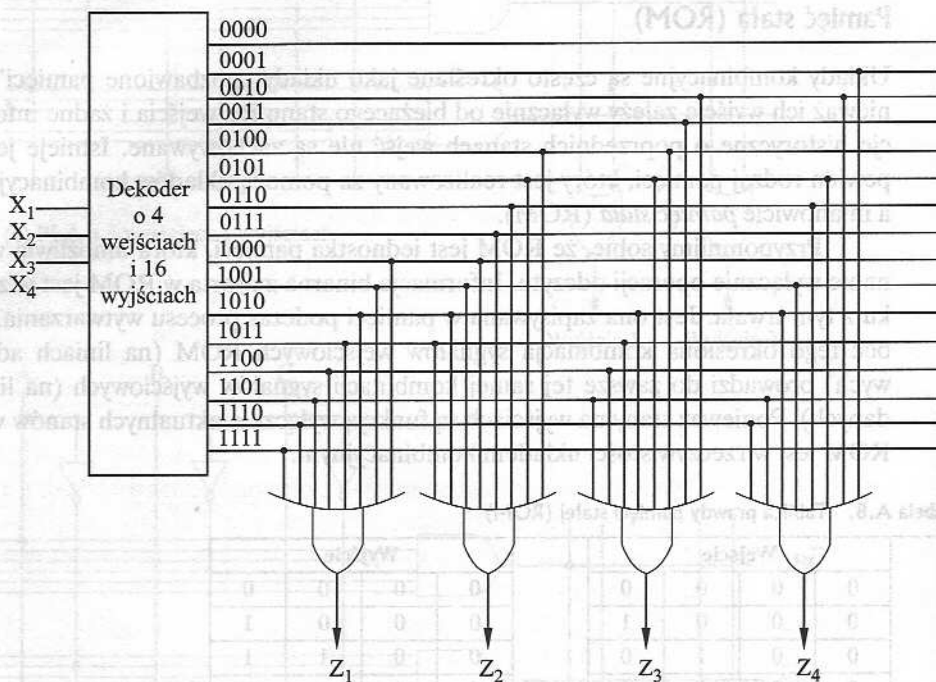
Układy kombinacyjne są często określane jako układy „pozbawione pamięci", ponieważ ich wyjście zależy wyłącznie od bieżącego stanu ich wejścia i żadne informacje historyczne o poprzednich stanach wejść nie są zachowywane. Istnieje jednak pewien rodzaj pamięci, który jest realizowany za pomocą układów kombinacyjnych, a mianowicie *pamięć stała* (ROM).

Przypomnijmy sobie, że ROM jest jednostką pamięci, która umożliwia wykonanie wyłącznie operacji odczytu. Informacja binarna zawarta w ROM jest w związku z tym trwała. Jest ona zapisywana w pamięci podczas procesu wytwarzania. Wobec tego określona kombinacja sygnałów wejściowych ROM (na liniach adresowych) prowadzi do zawsze tej samej kombinacji sygnałów wyjściowych (na liniach danych). Ponieważ stany na wyjściach są funkcją wyłącznie aktualnych stanów wejść, ROM jest w rzeczywistości układem kombinacyjnym.

Tabela A.8. Tablica prawdy pamięci stałej (ROM)

Wejście				Wyjście			
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Pamięć ROM może być wdrożona za pomocą dekodera i zbioru bramek OR. Jako przykład rozważmy tabelę A.8. Może ona być postrzegana jako tablica prawdy o 4 wejściach i 4 wyjściach. Dla każdej z 16 możliwych wartości wejściowych jest podany odpowiedni zbiór wartości wyjściowych. Tabelę tę można również widzieć jako definiującą zawartość 64-bitowej pamięci ROM składającej się z 16 słów 4-bitowych. Cztery wejścia określają adres, a 4 wyjścia – zawartość lokacji w pamięci określonej przez ten adres. Na rysunku A.20 jest pokazany sposób realizacji tej pamięci za pomocą dekodera 4 do 16 i 4 bramek OR. Podobnie jak w przypadku PLA, stosowana jest regularna organizacja, a schemat połączeń określa pożądaną zawartość.



Rysunek A.20. 64-bitowa pamięć ROM

Sumatory

Zobaczyliśmy dotychczas, w jaki sposób połączone bramki mogą być użyte do realizacji takich funkcji, jak kierowanie przepływem sygnałów, dekodowanie i pamięć ROM. Obszarem o zasadniczym znaczeniu, którego jeszcze nie rozpatrywaliśmy, są operacje arytmetyczne. W ramach tego krótkiego przeglądu zadowolimy się analizą funkcji dodawania.

Suma binarna różni się tym od sumy w algebrze Boole'a, że wynik obejmuje składnik przeniesienia. Wobec tego

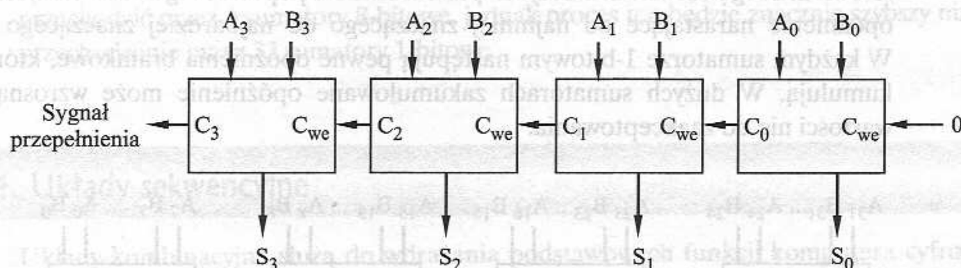
0	0	1	1
+0	+1	+0	+1
0	1	1	10

Mimo to dodawanie może być realizowane za pomocą wyrażeń Boole'a. W tabeli A.9a jest pokazana tablica prawdy dla sumowania dwóch bitów wejściowych w celu uzyskania 1-bitowej sumy i bitu przeniesienia. Tablica ta z łatwością może być zrealizowana w postaci cyfrowego układu logicznego. Nie jesteśmy jednak zainteresowani dodawaniem tylko pary bitów. Chcielibyśmy dodawać dwie liczby *n*-bitowe. Można to realizować przez zestawienie szeregu sumatorów w ten sposób, że przeniesienie z jednego sumatora jest doprowadzone na wejście następnego. Sumator 4-bitowy widać na rys. A.21.

Tabela A.9. Tablica prawdy dodawania binarnego

(a) Dodawanie pojedynczych bitów			
A	**B**	**Suma**	**Przeniesienie**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Dodawanie wraz z przeniesieniem				
C_{we}	**A**	**B**	**Suma**	C_{wy}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

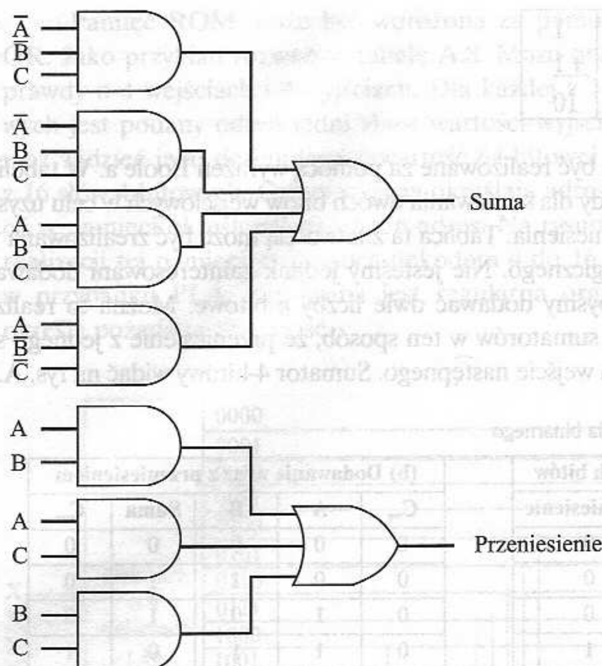


Rysunek A.21. Sumator 4-bitowy

Aby sumator wielobitowy mógł działać poprawnie, każdy z sumatorów 1-bitowych musi mieć 3 wejścia, włącznie z przeniesieniem z następnego sumatora niższego rzędu. Zrewidowana tablica prawdy znajduje się w tabeli A.9b. Stany dwóch wyjść mogą być wyrażone następująco:

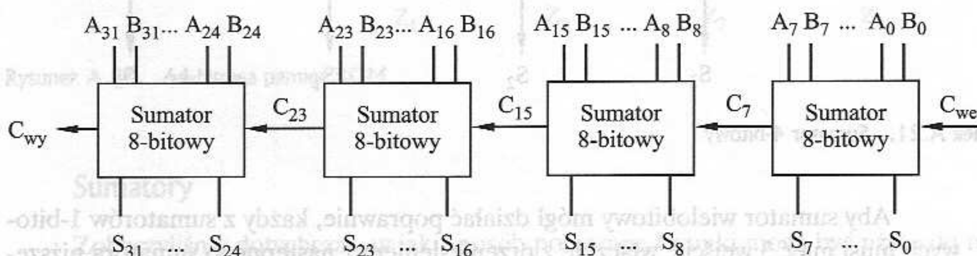
$$\text{Suma} = \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C}$$
$$\text{Przeniesienie} = AB + AC + BC$$

Na rysunku A.22 jest pokazana realizacja za pomocą bramek AND, OR i NOT.



Rysunek A.22. Realizacja sumatora

Dysponujemy więc układami logicznymi niezbędnymi do tego, by wdrożyć wielobitowy sumator, taki jak na rys. A.23. Zauważmy, że ponieważ wartość wyjściowa każdego sumatora zależy od przeniesienia z poprzedniego sumatora, istnieje opóźnienie narastające od najmniej znaczącego do najbardziej znaczącego bitu. W każdym sumatorze 1-bitowym następują pewne opóźnienia bramkowe, które się kumulują. W dużych sumatorach zakumulowane opóźnienie może wzrosnąć do wartości nie do zaakceptowania.



Rysunek A.23. Budowanie sumatora 32-bitowego za pomocą sumatorów 8-bitowych

Gdyby wartości przeniesienia mogły być określone bez przechodzenia przez wszystkie poprzednie stopnie, każdy sumator 1-bitowy mógłby działać niezależnie i opóźnienia by się nie kumulowały. Można to osiągnąć, stosując rozwiązanie nazy-

wane *układem przeniesienia na bardziej znaczące pozycje* (*carry lookahead*). W celu wyjaśnienia tego rozwiązania ponownie rozważmy sumator 4-bitowy.

Chcielibyśmy otrzymać wyrażenie, które określa wartość na wejściu przeniesienia dowolnego stopnia sumatora bez odnoszenia się do poprzednich wartości przeniesienia. Mamy więc

$$C_0 = A_0B_0 \tag{A.4}$$

$$C_1 = A_1B_1 + A_1A_0B_0 + B_1A_0B_0 \tag{A.5}$$

Powtarzając tę samą procedurę, otrzymujemy

$$C_2 = A_2B_2 + A_2A_1B_1 + A_2A_1A_0B_0 + A_2B_1A_0B_0 + B_2A_1B_1 + B_2A_1A_0B_0 + B_2B_1A_0B_0$$

Proces ten może być powtarzany dla dowolnie długich sumatorów. Każdy składnik przeniesienia może być wyrażony w postaci sumy iloczynów jako funkcja wyłącznie oryginalnych danych wejściowych, bez zależności od przeniesień. Wobec tego, niezależnie od długości sumatora, występują tylko dwa poziomy opóźnienia bramkowego.

W przypadku długich liczb rozwiązanie to staje się nadmiernie skomplikowane. Obliczanie wyrażenia dla najbardziej znaczącego bitu sumatora n-bitowego wymaga bramki OR o $n - 1$ wejściach oraz n bramek AND z liczbą wejść od 2 do $n + 1$. Dlatego też układ przeniesienia na bardziej znaczące pozycje jest zwykle realizowany jednocześnie dla 4÷8 bitów. Na rysunku A.23 jest pokazana budowa sumatora 32-bitowego złożonego z 4 sumatorów 8-bitowych. W tym przypadku przeniesienie musi przechodzić przez 4 sumatory 8-bitowe, jednak proces ten będzie znacznie szybszy niż przechodzenie przez 32 sumatory 1-bitowe.

A.4. Układy sekwencyjne

Układy kombinacyjne służą do wdrażania podstawowych funkcji komputera cyfrowego. Jednak, poza szczególnym przypadkiem pamięci ROM, nie umożliwiają one zrealizowania pamięci lub przechowywania informacji o stanie, które również mają zasadnicze znaczenie dla działania komputera. Do tego ostatniego celu służy bardziej złożona postać cyfrowych układów logicznych: układy sekwencyjne. Bieżący stan wyjścia układu sekwencyjnego zależy nie tylko od bieżącego stanu wejścia, ale również od historii stanu wejścia. Inny i na ogół bardziej użyteczny punkt widzenia jest taki, że bieżący stan wyjścia układu sekwencyjnego zależy od bieżącego wejścia oraz od bieżącego stanu samego układu.

W tym podrozdziale przeanalizujemy pewne proste, lecz użyteczne przykłady układów sekwencyjnych. Jak zobaczymy, w układzie sekwencyjnym wykorzystuje się układy kombinacyjne.

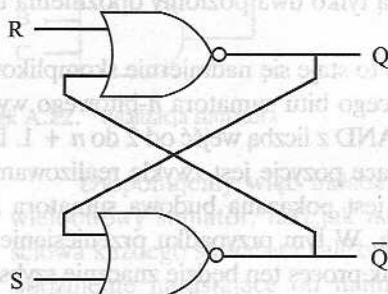
Przerzutniki

Najprostszą formą układu sekwencyjnego jest przerzutnik. Istnieją różne przerzutniki, jednak wszystkie mają następujące dwie właściwości:

- Przerzutnik jest układem dwustabilnym. Istnieje w jednym z dwóch stanów i bez sygnałów wejściowych zachowuje swój stan. Wobec tego przerzutnik może działać jako pamięć 1-bitowa.
- Przerzutnik ma dwa wyjścia, które zawsze dopełniają się wzajemnie. Są one zwykle oznaczane jako Q i $\overline{Q}$.

Przerzutnik zatrzaskowy S-R

Na rysunku A.24 widać powszechnie znaną konfigurację *przerzutnika S-R* zwanego też *zatrzaskiem* (przerzutnikiem zatrzaskowym) S-R. Układ ma dwa wejścia, S (*Set*, ustawienie) i R (*Reset*, przestawienie), oraz dwa wyjścia, Q i $\overline{Q}$. Składa się z dwóch bramek NOR połączonych ze sobą w układzie sprzężenia zwrotnego.



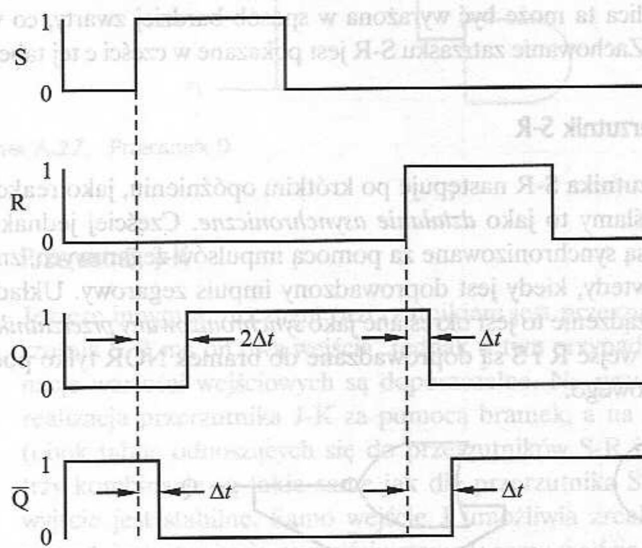
Rysunek A.24. Przerzutnik S-R typu zatrzask zrealizowany za pomocą bramek NOR

Pokażmy najpierw, że układ jest dwustabilny. Załóżmy, że S=0 i R=0 oraz że Q=0. Na wejściu dolnej bramki NOR są wartości Q=0 i S=0. Wobec tego stan wyjścia $\overline{Q}=1$ oznacza, że wartości na wejściach górnej bramki NOR są $\overline{Q}=1$ i R=0, co na wyjściu daje Q=0. Stan układu jest więc wewnętrznie spójny i pozostanie stabilny tak długo, jak długo S=R=0. Rozumując podobnie, dojdziemy do wniosku, że stan Q = 1, $\overline{Q}$ = 0 jest również stabilny przy R=S=0.

Układ może więc działać jako pamięć 1-bitowa. Możemy uważać wyjście Q za „wartość" bitu. Wejścia S i R służą do zapisu do pamięci wartości odpowiednio 1 i 0. Żeby to dostrzec, rozważmy stan Q=0, $\overline{Q}$ =1, S=0, R=0. Załóżmy, że stan S ulega zmianie na 1. Teraz na wejściach dolnej bramki NOR są wartości S=1, Q=0. Po pewnym opóźnieniu czasowym Δt na wyjściu dolnej bramki NOR będzie $\overline{Q}=0$ (patrz rys. A.25). Wobec tego wówczas na wejściach dolnej bramki NOR będą wartości R=0, Q=0. Po następnym opóźnieniu Δt wartość wyjścia Q stanie się równa 1. Jest to znowu stan stabilny. Na wejściach dolnej bramki są teraz wartości S=1, Q=1, a więc wyjście pozostaje w stanie Q=0. Tak długo, jak S=1 i R=0,

wartości wyjściowe pozostaną równe $Q = 1$, $\overline{Q} = 0$. Ponadto, jeśli S wróci do stanu 0, wartości na wyjściach pozostaną niezmienione.

Wejście R realizuje przeciwną funkcję. Gdy R przyjmuje stan 1, wymusza to $Q = 0$ i $\overline{Q} = 1$ niezależnie od poprzedniego stanu Q i $\overline{Q}$. Jak poprzednio, powtórna stabilizacja następuje po opóźnieniu $2\Delta t$ (rys. A.25).



Rysunek A.25. Przebiegi czasowe zatrzasku S-R zrealizowanego za pomocą bramek NOR

Tabela A.10. Przerzutnik S-R

(a) Tablica własności

Bieżące wejścia	Stan bieżący	Stan następny
SR	Q_n	Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	–
11	1	–

(b) Uproszczona tablica własności

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	–

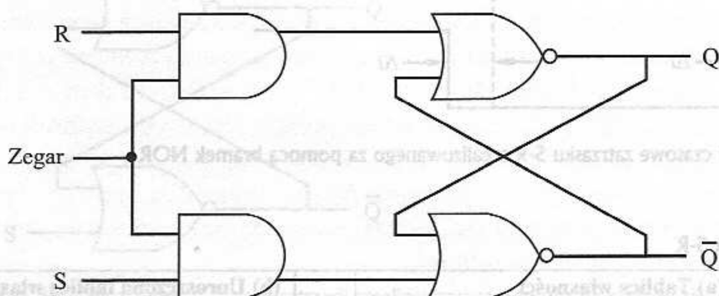
(c) Odpowiedź na szereg sygnałów wejściowych

t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
Q_{n+1}	1	1	1	0	0	0	0	0	1	1

Zatrzask S-R może być zdefiniowany za pomocą tablicy podobnej do tablicy prawdy, nazywanej *tablicą własności* (*characteristic table*), która pokazuje następny stan lub stany układu sekwencyjnego jako funkcję stanów bieżących i wartości wejściowych. W przypadku zatrzasku S-R stan może być zdefiniowany przez wartość Q. W tabeli A.10a pokazano wynikającą stąd tablicę własności. Zauważmy, że stan S=1, R=1 jest niedozwolony, ponieważ prowadziłoby to do niespójnego stanu wyjść (Q i $\overline{Q}$ jednocześnie równe 0). Tablica ta może być wyrażona w sposób bardziej zwarty, co widać w części b tabeli A.10. Zachowanie zatrzasku S-R jest pokazane w części c tej tabeli.

Synchronizowany przerzutnik S-R

Zmiana wyjścia przerzutnika S-R następuje po krótkim opóźnieniu, jako reakcja na zmianę wejścia. Określamy to jako *działanie asynchroniczne*. Częściej jednak zdarzenia w komputerze są synchronizowane za pomocą impulsów zegarowych, zmiany następują więc tylko wtedy, kiedy jest doprowadzony impuls zegarowy. Układ taki widać na rys. A.26. Urządzenie to jest określane jako *synchronizowany przerzutnik S-R*. Zauważmy, że stany z wejść R i S są doprowadzane do bramek NOR tylko podczas trwania impulsu zegarowego.



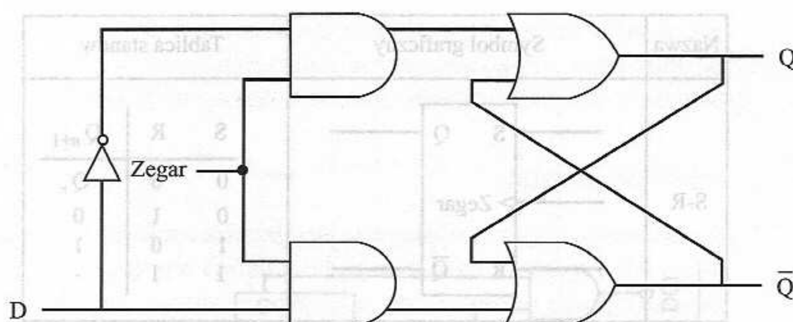
Rysunek A.26. Synchronizowany przerzutnik S-R

Przerzutnik D

W przypadku przerzutnika S-R problemem jest to, że należy zapobiegać powstaniu sytuacji, w której R = 1, S=1. Jednym ze sposobów osiągnięcia tego jest pozostawienie tylko jednego wejścia. Tak właśnie jest rozwiązany przerzutnik D. Na rysunku A.27 są pokazane realizacja za pomocą bramek oraz tablica własności przerzutnika D. Za pomocą inwertora zagwarantowano, że niezegarowe wejścia do dwóch bramek AND są swoimi przeciwieństwami.

Przerzutnik D jest czasami określany jako przerzutnik danych, ponieważ jest on w rzeczywistości komórką pamiętającą jeden bit danych. Stan na wyjściu przerzutnika D jest zawsze równy ostatniej wartości doprowadzonej na wejście. Przerzutnik pamięta więc ostatni sygnał wejściowy. Jest on również określany jako przerzutnik opóźniający, ponieważ pojawienie się na wejściu 0 lub 1 doprowadzonych do wejścia jest opóźnione o jeden impuls zegara.

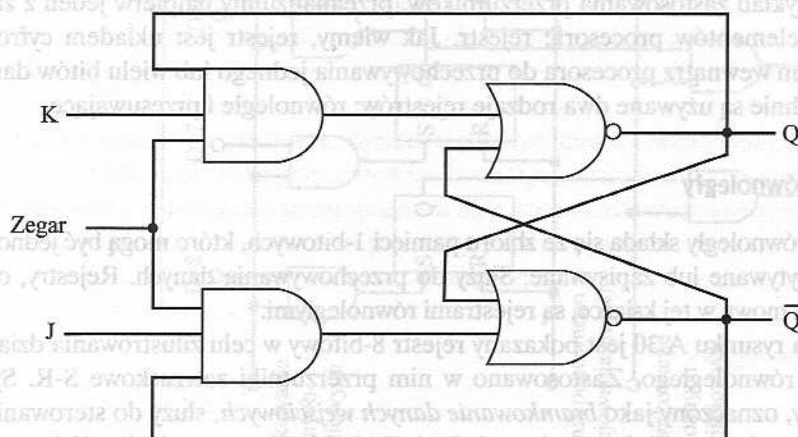
D	Q_{n+1}
0	0
1	1



Rysunek A.27. Przerzutnik D

Przerzutnik J-K

Jeszcze innym użytecznym przerzutnikiem jest przerzutnik J-K. Podobnie jak przerzutnik S-R ma on dwa wejścia. Jednak w tym przypadku wszystkie możliwe kombinacje wartości wejściowych są dopuszczalne. Na rysunku A.28 jest przedstawiona realizacja przerzutnika J-K za pomocą bramek, a na rys. A.29 – tablica własności (obok tablic odnoszących się do przerzutników S-R i D). Zauważmy, że pierwsze trzy kombinacje są takie same jak dla przerzutnika S-R. Bez sygnałów na wejściu wyjście jest stabilne. Samo wejście J umożliwia zrealizowanie funkcji ustawiania, powodując ustawienie na wejściu stanu 1; samo wejście K realizuje kasowanie (*reset*), powodując ustawienie na wyjściu stanu 0. Gdy oba wejścia (J i K) są równe 1, wykonywana funkcja jest określana jako kluczowanie: stan wyjścia ulega odwróceniu. Jeśli więc Q jest równe 1, a do wejść J i K zostaną doprowadzone 1, to Q stanie się równe 0. Czytelnik może zweryfikować, że wdrożenie przedstawione na rys. A.28 rzeczywiście realizuje tę charakterystyczną funkcję.



Rysunek A.28. Przerzutnik J-K

Nazwa	Symbol graficzny	Tablica stanów
S-R	S Q > Zegar R $\overline{Q}$	S R | Q_{n+1} 0 0 | Q_n 0 1 | 0 1 0 | 1 1 1 | -
J-K	J Q > Zegar K $\overline{Q}$	J K | Q_{n+1} 0 0 | Q_n 0 1 | 0 1 0 | 1 1 1 | $\overline{Q}_n$
D	D Q > Zegar $\overline{Q}$	D | Q_{n+1} 0 | 0 1 | 1

Rysunek A.29. Podstawowe przerzutniki

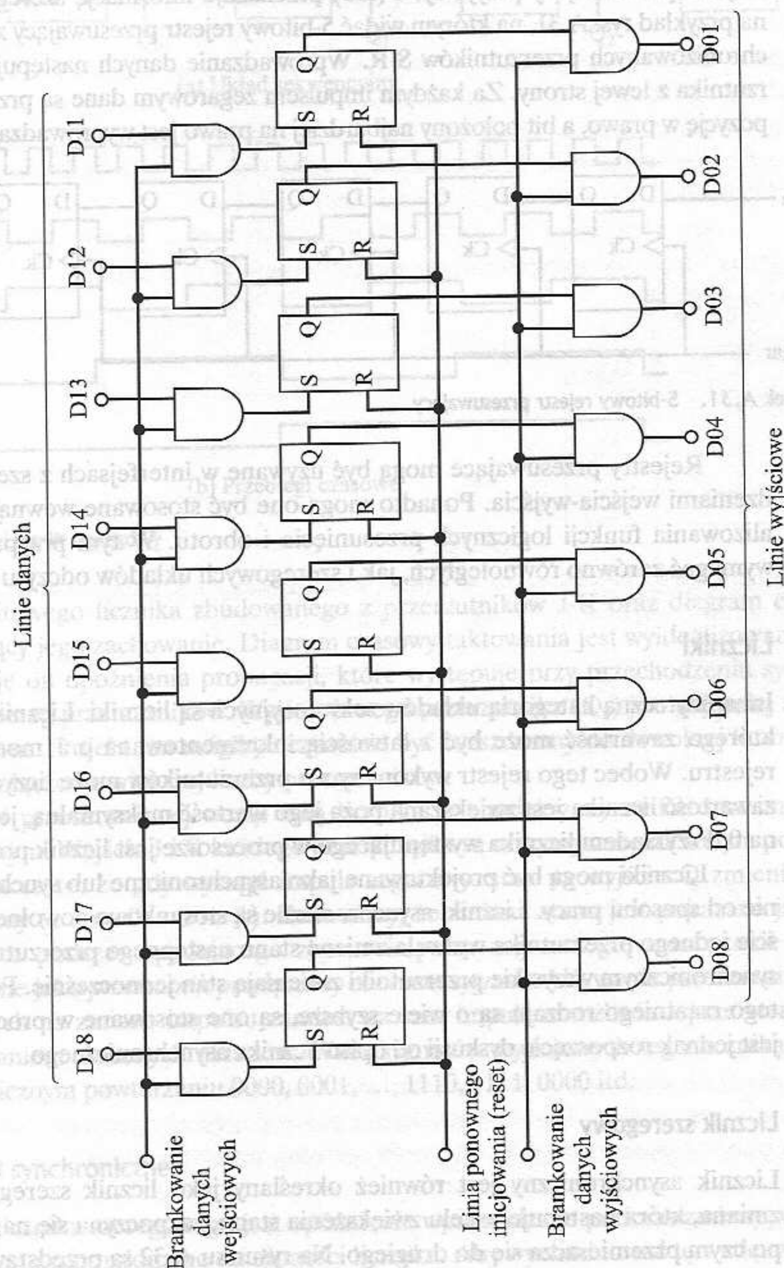
Rejestry

Jako przykład zastosowania przerzutników, przeanalizujmy najpierw jeden z zasadniczych elementów procesora: rejestr. Jak wiemy, rejestr jest układem cyfrowym używanym wewnątrz procesora do przechowywania jednego lub wielu bitów danych. Powszechnie są używane dwa rodzaje rejestrów: równoległe i przesuwające.

Rejestr równoległy

Rejestr równoległy składa się ze zbioru pamięci 1-bitowych, które mogą być jednocześnie odczytywane lub zapisywane. Służy do przechowywania danych. Rejestry, o których była mowa w tej książce, są rejestrami równoległymi.

Na rysunku A.30 jest pokazany rejestr 8-bitowy w celu zilustrowania działania rejestru równoległego. Zastosowano w nim przerzutniki zatrzaskowe S-R. Sygnał sterujący, oznaczony jako *bramkowanie danych wejściowych*, służy do sterowania zapisem do rejestru z linii sygnałowych D11÷D18. Linie te mogą być wyjściem multipleksera, co umożliwia ładowanie do rejestru danych z różnych źródeł. Wyjście jest

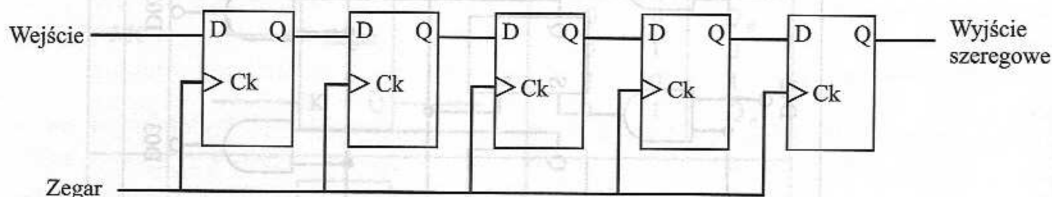


Rysunek A.30. 8-bitowy rejestr równoległy

sterowane w podobny sposób. Jako rozwiązanie dodatkowe przewidziano linię umożliwiającą łatwe zerowanie rejestru. Zauważmy, że nie mogłoby to tak łatwo być zrealizowane w przypadku rejestru zbudowanego z przerzutników D.

Rejestr przesuwający

Rejestr przesuwający przyjmuje i (lub) przekazuje informację szeregowo. Rozważmy na przykład rys. A.31, na którym widać 5-bitowy rejestr przesuwający zbudowany z synchronizowanych przerzutników S-R. Wprowadzanie danych następuje tylko do przerzutnika z lewej strony. Za każdym impulsem zegarowym dane są przesuwane o jedną pozycję w prawo, a bit położony najbardziej na prawo jest wyprowadzany na wyjście.



Rysunek A.31. 5-bitowy rejestr przesuwający

Rejestry przesuwające mogą być używane w interfejsach z szeregowymi urządzeniami wejścia-wyjścia. Ponadto mogą one być stosowane wewnątrz ALU do realizowania funkcji logicznych przesunięcia i obrotu. W tym przypadku mogą one wymagać zarówno równoległych, jak i szeregowych układów odczytu i zapisu.

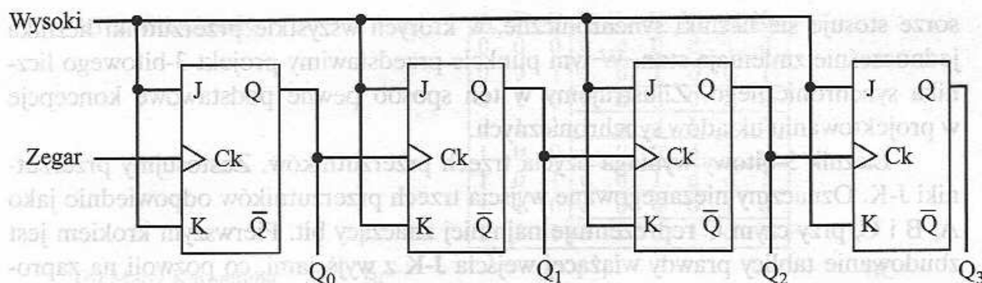
Liczniki

Inną użyteczną kategorią układów sekwencyjnych są liczniki. Licznik jest rejestrem, którego zawartość może być z łatwością inkrementowana o 1 modulo pojemność rejestru. Wobec tego rejestr wykonany z n przerzutników może liczyć do $2^n - 1$. Gdy zawartość licznika jest zwiększana poza jego wartość maksymalną, jest on ustawiany na 0. Przykładem licznika występującego w procesorze jest licznik programu.

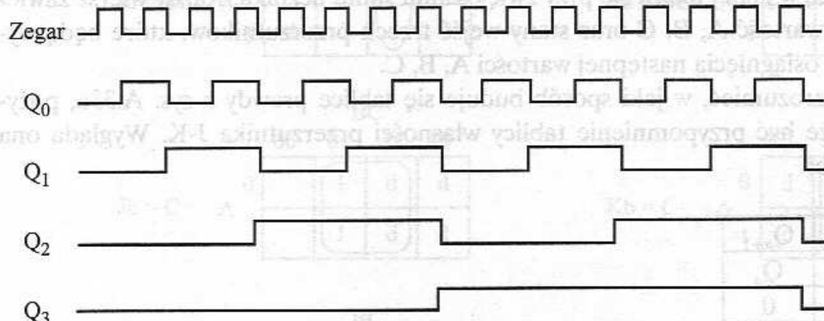
Liczniki mogą być projektowane jako asynchroniczne lub synchroniczne, zależnie od sposobu pracy. Liczniki asynchroniczne są stosunkowo powolne, ponieważ wyjście jednego przerzutnika wyzwala zmianę stanu następnego przerzutnika. W liczniku synchronicznym wszystkie przerzutniki zmieniają stan jednocześnie. Ponieważ liczniki tego ostatniego rodzaju są o wiele szybsze, są one stosowane w procesorze. Celowe jest jednak rozpoczęcie dyskusji od opisu licznika asynchronicznego.

Licznik szeregowy

Licznik asynchroniczny jest również określany jako licznik szeregowy, ponieważ zmiana, która następuje w celu zwiększenia stanu, rozpoczyna się na jednym końcu, po czym przemieszcza się do drugiego. Na rysunku A.32 są przedstawione: wdroże-



(a) Układ sekwencyjny



(b) Przebiegi czasowe

Rysunek A.32. Licznik szeregowy

nie 4-bitowego licznika zbudowanego z przerzutników J-K oraz diagram czasowy ilustrujący jego zachowanie. Diagram czasowy taktowania jest wyidealizowany – nie pokazuje on opóźnienia propagacji, które występuje przy przechodzeniu sygnałów przez szereg przerzutników. Wyjście lewego przerzutnika (Q_0) jest najmniej znaczącym bitem. Projekt ten mógłby oczywiście być rozszerzony do dowolnej liczby bitów przez dołączenie większej liczby przerzutników.

W przedstawionej wersji stan licznika jest zwiększany za każdym impulsem zegarowym. Wejścia J i K każdego przerzutnika są utrzymywane na stałym poziomie 1. Oznacza to, że gdy wystąpi impuls zegarowy, stan na wyjściu Q zmieni się na przeciwny (z 1 na 0; z 0 na 1). Zauważmy, że zmiana stanu jest pokazana jako następująca podczas opadającego zbocza impulsu zegarowego; rozwiązanie to jest określane jako *przerzutnik przełączany zboczem sygnału*. Używanie przerzutników reagujących na zmianę impulsu zamiast na sam impuls umożliwia lepsze sterowanie taktowaniem złożonych układów. Wzór sygnałów wyjściowych tego licznika polega na cyklicznym powtarzaniu 0000, 0001, ..., 1110, 1111, 0000 itd.

Liczniki synchroniczne

Wadą licznika szeregowego jest opóźnienie wprowadzane podczas zmiany wartości, które jest proporcjonalne do długości licznika. Aby wyeliminować tę wadę, w proce-

sorze stosuje się liczniki synchroniczne, w których wszystkie przerzutniki licznika jednocześnie zmieniają stan. W tym punkcie przedstawimy projekt 3-bitowego licznika synchronicznego. Zilustrujemy w ten sposób pewne podstawowe koncepcje w projektowaniu układów synchronicznych.

Licznik 3-bitowy wymaga użycia trzech przerzutników. Zastosujmy przerzutniki J-K. Oznaczmy niezanegowane wyjścia trzech przerzutników odpowiednio jako A, B i C, przy czym C reprezentuje najmniej znaczący bit. Pierwszym krokiem jest zbudowanie tablicy prawdy wiążącej wejścia J-K z wyjściami, co pozwoli na zaprojektowanie całego układu. Tablica taka jest pokazana na rys. A.33a. W pierwszych trzech kolumnach są podane możliwe kombinacje wyjść A, B i C. Wymieniono je w kolejności, w jakiej ukażą się przy zwiększaniu stanu licznika. Każdy wiersz zawiera bieżącą wartość A, B, C oraz stany wejść trzech przerzutników, które będą wymagane do osiągnięcia następnej wartości A, B, C.

Aby zrozumieć, w jaki sposób buduje się tablice prawdy z rys. A.33a, pożyteczne może być przypomnienie tablicy własności przerzutnika J-K. Wygląda ona następująco:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

W tej postaci tablica ukazuje wpływ, jaki wejścia J i K mają na wyjście. Rozważmy teraz następującą organizację tych samych informacji:

Q_n	J	K	Q_{n+1}
0	0	d	0
0	1	d	1
1	d	1	0
1	d	0	1

W tej postaci tablica określa następny stan wyjścia, gdy są znane wejścia i aktualne wyjście. Jest to dokładnie taka informacja, jaka jest potrzebna do projektowania licznika lub w istocie dowolnego układu sekwencyjnego. W tej postaci tablica jest określana jako *tablica wzbudzania*.

Powróćmy do rysunku A.33a. Rozważmy pierwszy wiersz. Chcemy, żeby wartości A i B pozostały równe 0, a wartość C zmieniła się z 0 na 1 przy następnym impulsie zegarowym. Tablica wzbudzania pokazuje, że aby zachować wartość wyjścia 0, musimy mieć na wejściu J = 0 oraz dowolną wartość na wejściu K. Aby nastąpiła zmiana z 0 na 1, wejścia muszą być równe J = 1 i K = d. Wartości te są pokazane w pierwszym wierszu tablicy. Drogą podobnego rozumowania można wypełnić pozostałą część tablicy.

(a) Tablica prawdy

A	B	C	Ja	Ka	Jb	Kb	Jc	Kc
0	0	0	0	d	0	d	1	d
0	0	1	0	d	1	d	d	1
0	1	0	0	d	d	0	1	d
0	1	1	1	d	d	1	d	1
1	0	0	d	0	0	d	1	d
1	0	1	d	0	1	d	d	1
1	1	0	d	0	d	0	1	d
1	1	1	d	1	d	1	d	1

(b) Mapy Karnaugha

Ja = BC

A \ BC	00	01	11	10
0			1	
1	d	d	d	d

Ka = BC

A \ BC	00	01	11	10
0	d	d	d	d
1			1	

Jb = C

A \ BC	00	01	11	10
0		1	d	d
1		1	d	d

Kb = C

A \ BC	00	01	11	10
0	d	d	1	
1	d	d	1	

Jc = 1

A \ BC	00	01	11	10
0	1	d	d	1
1	1	d	d	1

Kc = 1

A \ BC	00	01	11	10
0	d	1	1	d
1	d	1	1	d

(c) Schemat logiczny

Wysoki
J C
Ck
K $\overline{C}$
J B
Ck
K $\overline{B}$
J A
Ck
K $\overline{A}$
Zegar

Rysunek A.33. Projektowanie licznika synchronicznego

Po utworzeniu tablicy prawdy z rys. A.33a widzimy, że tablica ta pokazuje wymagane wartości wszystkich wejść J i K jako funkcje bieżących wartości A, B i C. Za pomocą map Karnaugha możemy znaleźć wyrażenia Boole'a dla tych sześciu funkcji. Pokazano to w części b rysunku. Na przykład mapa Karnaugha dla zmiennej Ja (wejście J przerzutnika, który daje na wyjściu wartość A) prowadzi do wyrażenia Ja = BC. Po wyprowadzeniu wszystkich sześciu wyrażeń zaprojektowanie układu jest rzeczą prostą, co widać w części c rysunku.

A.5. Problemy do rozwiązania

A.1. Zbuduj tablice prawdy dla następujących wyrażeń Boole'a:

(a) $ABC+\overline{ABC}$ (b) $ABC+A\overline{BC}+\overline{ABC}$

(c) $A(B\overline{C}+\overline{B}C)$ (d) $(A+B)(A+C)(\overline{A}+\overline{B})$

A.2. Uprość następujące wyrażenia zgodnie z prawem przemienności:

(a) $A\cdot\overline{B}+\overline{B}\cdot A+C\cdot D\cdot E+\overline{C}\cdot D\cdot E+E\cdot\overline{C}\cdot D$

(b) $A\cdot B+A\cdot C+B\cdot A$

(c) $(L\cdot M\cdot N)(A\cdot B)(C\cdot D\cdot E)(M\cdot N\cdot L)$

(d) $F\cdot(K+R)+S\cdot V+W\cdot\overline{X}+V\cdot S+\overline{X}\cdot W+(R+K)\cdot F$

A.3. Zastosuj twierdzenie De Morgana do następujących równań:

(a) $F=\overline{V+A+L}$

(b) $F=\overline{A}+\overline{B}+\overline{C}+\overline{D}$

A.4. Uprość następujące wyrażenia:

(a) $A=S\cdot T+V\cdot W+R\cdot S\cdot T$

(b) $A=T\cdot U\cdot V+X\cdot Y+Y$

(c) $A=F\cdot(E+F+G)$

(d) $A=(P\cdot Q+R+S\cdot T)T\cdot S$

(e) $A=\overline{D}\cdot\overline{D}\cdot E$

(f) $A=Y\cdot(W+X+\overline{\overline{\overline{Y}}+\overline{\overline{Z}}})\cdot Z$

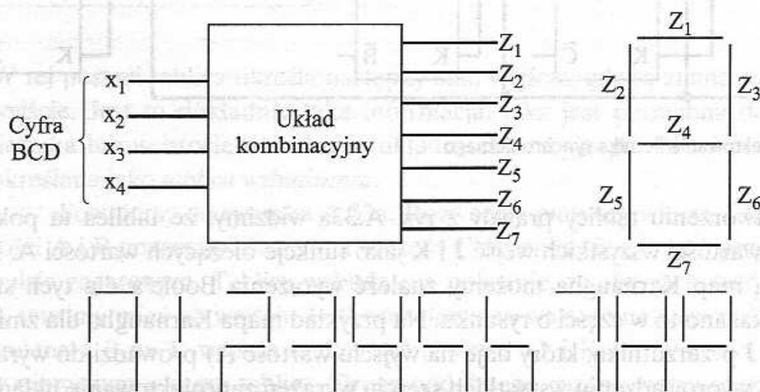
(g) $A=(B\cdot E+C+F)\cdot C$

A.5. Zbuduj operację XOR z podstawowych operacji Boole'a AND, OR i NOT.

A.6. Dysponując bramką NOR i bramkami NOT, narysuj schemat logiczny układu realizującego funkcję AND z 3 wejściami.

A.7. Napisz wyrażenie Boole'a dla 4-wejściowej bramki NAND.

A.8. Do sterowania 7-segmentowym wskaźnikiem cyfr dziesiętnych jest używany układ kombinacyjny, co pokazano na rys. A.34. Układ ma 4 wejścia doprowadzające kod 4-bitowy używany w upakowanej reprezentacji dziesiętnej ($0_{10}=0000$, ..., $9_{10}=1001$). Siedem wyjść określa, które segmenty są wzbudzane w celu wyświetlenia danej cyfry dziesiętnej. Zauważ, że niektóre kombinacje wejść i wyjść są niepotrzebne.



Rysunek A.34. Przykład 7-segmentowego wyświetlacza na diodach świecących (LED)

(a) Zbuduj tablicę prawdy dla tego układu.
(b) Wyraź tablicę prawdy w formie sumy iloczynów.
(c) Wyraź tablicę prawdy w formie iloczynu sum.
(d) Wyprowadź wyrażenie uproszczone.

A.9. Zaprojektuj multiplekser 8 do 1.

A.10. Dodaj linię do schematu przedstawionego na rys. A.15 tak, żeby układ ten pełnił rolę demultipleksera.

A.11. Kod Graya jest kodem binarnym liczb całkowitych. Różni się on od zwykłej reprezentacji binarnej tym, że między reprezentacjami dwóch dowolnych liczb istnieje tylko różnica jednego bitu*. Jest to przydatne w takich zastosowaniach, jak liczniki lub konwertery analogowo-cyfrowe, w których jest generowana sekwencja liczb. Ponieważ w określonej chwili ulega zmianie tylko jeden bit, nigdy nie występuje niejednoznaczność spowodowana przez niewielkie różnice taktowania. Oto pierwszych 8 elementów kodu:

Kod binarny	**Kod Graya**
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Zaprojektuj układ, który przekształca kod binarny na kod Graya.

A.12. Zaprojektuj dekoder 5 × 32, używając 4 dekoderów 3 × 8 (z wejściami zezwolenia) i jednego dekodera 2 × 4.

A.13. Opracuj implementację pełnego sumatora z rys. A.22, używając tylko 5 bramek (*wskazówka*: niektóre z tych bramek to bramki XOR).

A.14. Rozważ rysunek A.22. Załóż, że każda bramka powoduje opóźnienie 10 ns. Wobec tego wyjście sumatora jest ważne po 30 ns, a wyjście przeniesienia po 0 ns. Jaki jest całkowity czas operacji dodawania sumatora 32-bitowego zrealizowanego:
(a) bez układu przeniesienia na bardziej znaczące pozycje, jak na rys. A.21?
(b) z układem przeniesienia na bardziej znaczące pozycje i przy użyciu sumatorów 8-bitowych, jak na rys. A.23?

* Chodzi o kolejne liczby (*przyp. tłum.*).

(a) Zbuduj tablicę prawdy dla tego układu.
(b) Wyraź tablicę prawdy w formie sumy iloczynów.
(c) Wyraź tablicę prawdy w formie iloczynu sum.
(d) Wyprowadź wyrażenie uproszczone.

A.9. Zaprojektuj multiplekser 8 do 1.

A.10. Dodaj linie do schematu przedstawionego na rys. A.15 tak, żeby układ ten pełnił rolę demultipleksera.

A.11. Kod Graya jest kodem binarnym liczb całkowitych. Różni się on od zwykłej reprezentacji binarnej tym, że między reprezentacjami dwóch dowolnych liczb istnieje tylko różnica jednego bitu*. Jest to przydatne w takich zastosowaniach, jak liczniki lub konwertery analogowo-cyfrowe, w których jest generowana sekwencja liczb. Ponieważ w określonej chwili ulega zmianie tylko jeden bit, nigdy nie występuje niejednoznaczność spowodowana przez niewielkie różnice taktowania. Oto pierwszych 8 elementów w kodzie:

Kod binarny	Kod Graya
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Zaprojektuj układ, który przekształca kod binarny na kod Graya.

A.12. Zaprojektuj dekoder 5 × 32, używając 4 dekoderów 3 × 8 (z wejściami zezwolenia) i jednego dekodera 2 × 4.

A.13. Opracuj implementację pełnego sumatora z rys. A.22, używając tylko 5 bramek (wskazówka: niektóre z tych bramek to bramki XOR).

A.14. Rozważ rysunek A.22. Załóż, że każda bramka powoduje opóźnienie 10 ns. Wobec tego wyjście sumatora jest ważne po 30 ns, a wyjście przeniesienia po 20 ns. Jaki jest całkowity czas operacji dodawania sumatora 32-bitowego zrealizowanego:
(a) bez układu przeniesienia na bardziej znaczące pozycje, jak na rys. A.21?
(b) z układem przeniesienia na bardziej znaczące pozycje i przy użyciu sumatorów 8-bitowych, jak na rys. A.23?

* Chodzi o kolejne liczby (przyp. tłum.).

Dodatek B
Systemy liczbowe

B.1. System dziesiętny
B.2. System binarny
B.3. Konwersja między liczbami dziesiętnymi a binarnymi
Liczby całkowite
Ułamki
B.4. Notacja szesnastkowa
B.5. Problemy do rozwiązania

B.1. System dziesiętny

W codziennym życiu do reprezentowania liczb używamy systemu opartego na cyfrach dziesiętnych (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), a sam system określamy jako *dziesiętny* lub *dziesiątkowy*. Zastanówmy się, co oznacza liczba 83. Zapis ten należy rozumieć jako 8 dziesiątek plus 3:

$$83 = 8 \times 10 + 3$$

Liczba 4728 oznacza 4 tysiące, 7 setek, 2 dziesiątki plus 8:

$$4728 = 4 \times 1000 + 7 \times 100 + 2 \times 10 + 8$$

Mówi się, że system dziesiętny ma **podstawę** 10. Oznacza to, że każda cyfra w liczbie jest mnożona przez 10 do potęgi odpowiadającej pozycji tej cyfry. Wobec tego

$$83 = (8 \times 10^1) + (3 \times 10^0)$$

$$4728 = (4 \times 10^3) + (7 \times 10^2) + (2 \times 10^1) + (8 \times 10^0)$$

Wartości ułamkowe są reprezentowane zgodnie z tą samą zasadą, jednak są używane ujemne potęgi 10. Zatem ułamek dziesiętny 0,256 to 2 dziesiąte plus 5 setnych plus 6 tysięcznych:

$$0{,}256 = (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

Cyfry liczby zawierającej część całkowitą i ułamkową są mnożone przez 10 podnoszone zarówno do potęgi dodatniej, jak i ujemnej:

$$472{,}256 = (4 \times 10^2) + (7 \times 10^1) + (2 \times 10^0) + (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

Ogólnie, dziesiętna reprezentacja $X = \{\ldots d_2 d_1 d_0, d_{-1} d_{-2} d_{-3} \ldots\}$, a wartość X można zapisać wzorem:

$$X = \sum_i d_i 10^i$$

B.2. System binarny

W systemie dziesiętnym używa się dziesięciu różnych cyfr do reprezentowania liczb o podstawie 10. W systemie binarnym dysponujemy tylko dwiema cyframi, 1 i 0. Wobec tego liczby w systemie binarnym są reprezentowane przy podstawie 2.

Aby zapobiec nieporozumieniom, stosujemy czasem przy liczbach indeksy wskazujące ich podstawę. Na przykład, 83_{10} i 4728_{10} są liczbami reprezentowanymi w notacji dziesiętnej lub po prostu dziesiętnymi. W notacji binarnej 1 i 0 mają to samo znaczenie co w notacji dziesiętnej:

$$0_2 = 0_{10}$$

$$1_2 = 1_{10}$$

Jak reprezentujemy większe liczby? Podobnie jak w przypadku notacji dziesiętnej, każda cyfra w liczbie binarnej ma wartość zależną od jej pozycji:

$$10_2 = (1 \times 2^1) + (0 \times 2^0) = 2_{10}$$

$$11_2 = (1 \times 2^1) + (1 \times 2^0) = 3_{10}$$

$$100_2 = (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4_{10}$$

i tak dalej. Jak poprzednio, wartości ułamkowe są reprezentowane za pomocą ujemnych potęg podstawy:

$$1001{,}101 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9{,}625_{10}$$

Ogólnie reprezentacja binarna $Y = \{\ldots b_2 b_1 b_0 b_{-1} b_{-2} b_{-3} \ldots\}$, a jej wartością jest

$$Y = \sum_i b_i \times 2^i$$

B.3. Konwersja między liczbami dziesiętnymi a binarnymi

Konwersja liczby z notacji binarnej na dziesiętną jest prosta. W poprzednim punkcie pokazaliśmy kilka przykładów takiej konwersji. Wszystko co jest potrzebne, to pomnożenie każdej cyfry binarnej przez odpowiednią potęgę 2 i dodanie wyników.

W przypadku konwersji z notacji dziesiętnej na binarną część całkowita i część ułamkowa są traktowane oddzielnie.

Liczby całkowite

Pamiętamy, że w notacji binarnej liczba całkowita reprezentowana przez

$$b_{m-1} b_{m-2} \ldots b_2 b_1 b_0 \qquad b_i = 0 \text{ lub } 1$$

ma wartość

$$(b_{m-1} \times 2^{m-1}) + (b_{m-2} \times 2^{m-2}) + \ldots + (b_1 \times 2^1) + b_0$$

Załóżmy, że jest wymagana konwersja dziesiętnej liczby całkowitej N na postać binarną. Jeśli podzielimy N przez 2 (w systemie dziesiętnym) i otrzymamy iloraz N_1 i resztę R_0, to możemy zapisać, że

$$N = 2 \times N_1 + R_0 \qquad R_0 = 0 \text{ lub } 1$$

Następnie dzielimy iloraz N_1 przez 2. Nowy iloraz niech wynosi N_2, a nowa reszta – R_1. Wobec tego

$$N_1 = 2 \times N_2 + R_1 \qquad R_1 = 0 \text{ lub } 1$$

stąd

$$N = 2\,(2N_2 + R_1) + R_0 = (N_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

Jeśli następnie

$$N_2 = 2N_3 + R_2$$

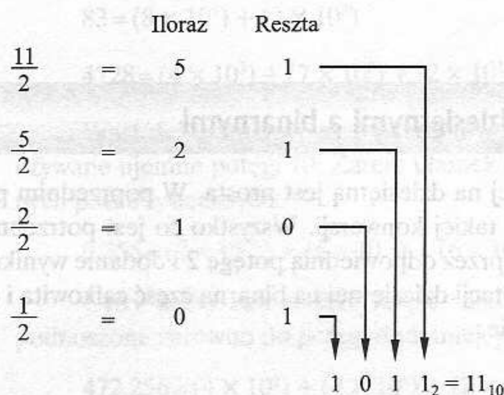
to mamy

$$N = (N_3 \times 2^3) + (R_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

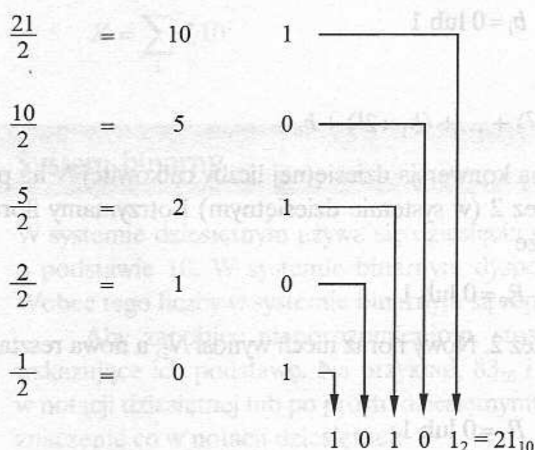
Kontynuując to, ze względu na $N > N_1 > N_2 \ldots$, możemy w końcu otrzymać iloraz $N_{m-1} = 1$ oraz resztę R_{m-2}, która jest równa 0 lub 1. Wobec tego

$$N = (1 \times 2^{m-1}) + (R_{m-2} \times 2^{m-2}) + \ldots + (R_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

co jest binarną postacią N. Oznacza to, że konwersji z podstawy 10 do podstawy 2 dokonuje się, powtarzając dzielenie przez 2. Kolejne reszty oraz ostatni iloraz (1) dają nam cyfry binarne N w kolejności rosnącego znaczenia. Na rysunku B.1 są pokazane dwa przykłady.



(a) 11_{10}



(b) 21_{10}

Rysunek B.1. Przykłady konwersji liczb całkowitych z notacji dziesiętnej na binarną

Ułamki

Pamiętamy, że w notacji binarnej liczba o wartości między 0 a 1 jest reprezentowana przez

$$0,b_{-1}b_{-2}b_{-3}\ldots \qquad b_i = 0 \text{ lub } 1$$

i ma wartość

$$(b_{-1}\times 2^{-1}) + (b_{-2}\times 2^{-2}) + (b_{-3}\times 2^{-3})\ldots$$

co można zapisać jako

$$2^{-1}\times(b_{-1} + 2^{-1}\times(b_{-2} + 2^{-1}\times(b_{-3} + \ldots$$

Wyrażenie to sugeruje nam sposób konwersji. Załóżmy, że należy dokonać konwersji liczby F $(0 < F < 1)$ z notacji dziesiętnej na binarną. Wiemy, że F możemy wyrazić jako

$$F = 2^{-1}\times(b_{-1} + 2^{-1}\times(b_{-2} + 2^{-1}\times(b_{-3} + \ldots$$

Jeśli pomnożymy F przez 2, otrzymamy

$$2\times F = b_{-1} + 2^{-1}\times(b_{-2} + 2^{-1}\times(b_{-3} + \ldots$$

Iloczyn	Część całkowita
0.81 × 2 = 1.62	1
0.62 × 2 = 1.24	1
0.24 × 2 = 0.48	0
0.48 × 2 = 0.96	0
0.96 × 2 = 1.92	1
0.92 × 2 = 1.84	1

.1 1 0 0 1 1

(a) $0.81_{10} = 0.110011_2$ (w przybliżeniu)

Iloczyn	Część całkowita
0.25 × 2 = 0.5	0
0.5 × 2 = 1.0	1

.0 1

(b) $0.25_{10} = 0.01_2$ (dokładnie)

Rysunek B.2. Przykłady konwersji liczb ułamkowych z notacji dziesiętnej na binarną

Na podstawie tego równania widzimy, że część całkowita $2F$, która musi być albo 0, albo 1 (ponieważ $0 < F < 1$), jest równa po prostu b_{-1}. Stąd $2F = b_{-1} + F_1$, gdzie $0 < F_1 < 1$, oraz

$$F_1 = 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + 2^{-1} \times (b_{-4} + \ldots$$

W celu znalezienia b_{-2} powtarzamy ten proces. Algorytm konwersji obejmuje więc powtarzane mnożenie przez 2. Na każdym etapie ułamkowa część liczby z poprzedniego etapu jest mnożona przez 2. Cyfra znajdująca się po lewej stronie przecinka w iloczynie jest równa 0 lub 1 i wnosi wkład do reprezentacji binarnej, poczynając od cyfry najbardziej znaczącej. Część ułamkowa iloczynu służy jako mnożna w następnym etapie. Dwa przykłady zostały pokazane na rys. B.2.

Proces ten niekoniecznie jest dokładny; chodzi o to, że ułamek dziesiętny o skończonej liczbie cyfr może wymagać ułamka binarnego o nieskończonej liczbie cyfr. W takich przypadkach algorytm konwersji jest zwykle zatrzymywany po wstępnie ustalonej liczbie kroków, zależnie od wymaganej dokładności.

B.4. Notacja szesnastkowa

Ze względu na nieodłączną, binarną naturę zespołów komputera cyfrowego, wszystkie formy danych wewnątrz komputera są reprezentowane za pomocą różnych kodów binarnych. Widzieliśmy przykłady kodów binarnych dla tekstu i notacji binarnej dla liczb. Później zobaczymy przykłady użycia kodów binarnych dla innych rodzajów danych. Niezależnie jednak od tego, jak wygodny jest system binarny z punktu widzenia komputerów, jest on zbyt nieporęczny z punktu widzenia ludzi. W związku z tym wielu specjalistów, którzy muszą spędzać czas, pracując z danymi pierwotnymi w komputerach, woli bardziej zwartą notację.

Jakiej użyć notacji? Jedną z możliwości jest notacja dziesiętna. Z pewnością notacja ta jest bardziej zwarta niż binarna, jednak jest kłopotliwa z powodu żmudnego procesu konwersji między podstawą 2 a podstawą 10.

Zamiast tego przyjęto notację *szesnastkową*. Cyfry binarne są grupowane w zespoły po 4. Każda możliwa kombinacja 4 cyfr binarnych otrzymuje następujący symbol:

0000=0	1000=8
0001=1	1001=9
0010=2	1010=A
0011=3	1011=B
0100=4	1100=C
0101=5	1101=D
0110=6	1110=E
0111=7	1111=F

Ponieważ użyto 16 symboli, notacja została nazwana *szesnastkową*, a 16 symboli – *cyframi szesnastkowymi*.

Ciąg cyfr szesnastkowych może być traktowany jako reprezentacja liczby całkowitej o podstawie 16. Wobec tego

$$2C_{16} = (2_{16} \times 16^1) + (C_{16} \times 16^0) = (2_{10} \times 16^1) + (12_{10} \times 16^0) = 44$$

Jednak notacja szesnastkowa jest używana nie tylko do reprezentowania liczb całkowitych. Jest stosowana jako zwięzła notacja do reprezentowania dowolnego ciągu cyfr binarnych, niezależnie od tego, czy reprezentują one tekst, liczby czy inny rodzaj danych. Przyczyny stosowania notacji szesnastkowej są następujące:

1. Jest bardziej zwarta niż notacja binarna.
2. W większości komputerów dane binarne zajmują pewne wielokrotności 4 bitów, co oznacza wielokrotność pojedynczych cyfr szesnastkowych.
3. Konwersja między notacją binarną a szesnastkową jest bardzo łatwa.

Jako przykład dotyczący ostatniego punktu rozważmy ciąg binarny 110111100001. Jest on równoważny następującemu zapisowi szesnastkowemu:

1101	1110	0001	$= DE1_{16}$
D	E	1	

Proces ten jest przeprowadzany tak naturalnie, że doświadczony programista może w myśli przetwarzać reprezentacje binarne na szesnastkowe bez konieczności pisania.

B.5. Problemy do rozwiązania

B.1. Przekształć następujące liczby binarne na ich równoważniki dziesiętne:
(a) 001100 (b) 000011 (c) 011100 (d) 111100 (e) 101010

B.2. Przekształć następujące liczby binarne na ich równoważniki dziesiętne:
(a) 11100,011 (b) 110011,10011 (c) 1010101010,1

B.3. Przekształć następujące liczby dziesiętne na ich równoważniki binarne:
(a) 64 (b) 100 (c) 111 (d) 145 (e) 255

B.4. Przekształć następujące liczby dziesiętne na ich równoważniki binarne:
(a) 34,75 (b) 25,25 (c) 27,1875

B.5. Wyraź następujące liczby ósemkowe w notacji szesnastkowej:
(a) 12 (b) 5655 (c) 2550276 (d) 76545336 (e) 37267555

B.6. Przekształć następujące liczby szesnastkowe na ich równoważniki dziesiętne:
(a) C (b) 9F (c) D52 (d) 67E (e) ABCD

B.7. Przekształć następujące liczby szesnastkowe na ich równoważniki dziesiętne:
(a) F,4 (b) D3,E (c) 1111,1 (d) 888,8 (e) EBA,C

B.8. Przekształć następujące liczby dziesiętne na ich równoważniki szesnastkowe:

(a) 16 (b) 80 (c) 2560 (d) 3000 (e) 62500

B.9. Przekształć następujące liczby dziesiętne na ich równoważniki szesnastkowe:

(a) 204,125 (b) 255,875 (c) 631,25 (d) 10000,00390625

B.10. Przekształć następujące liczby szesnastkowe na ich równoważniki binarne:

(a) E (b) 1C (c) A64 (d) 1F,C (e) 239,4

B.11. Przekształć następujące liczby binarne na ich równoważniki szesnastkowe:

(a) 1001,1111 (b) 110101,011001 (c) 10100111,111011

B.12. Udowodnij, że każda liczba rzeczywista o skończonej reprezentacji binarnej (tzn. o skończonej liczbie cyfr po przecinku) ma również skończoną reprezentację dziesiętną.

Dodatek C

Zadania ułatwiające nauczanie organizacji i architektury komputera

C.1. Wyszukiwanie informacji
C.2. Zadania symulacyjne
SimpleScalar
SMPCache
C.3. Zadania typu lektura-sprawozdanie

Wielu wykładowców jest przekonanych, że wyszukiwanie informacji lub przedsięwzięcia symulacyjne mają zasadnicze znaczenie dla jasnego zrozumienia koncepcji organizacji i architektury komputera. Bez takich przedsięwzięć trudno jest studentom pojąć niektóre podstawowe koncepcje i wzajemne oddziaływanie między poszczególnymi składnikami. Przedsięwzięcia umacniają koncepcje przedstawiane w książce, umożliwiają studentom lepsze zrozumienie wewnętrznego funkcjonowania procesora, a także mogą lepiej umotywować studentów i wpoić w nich przekonanie, że opanowali niezbędny materiał.

W tej książce próbowałem przedstawić koncepcje możliwie jasno i w celu ich umocnienia wprowadziłem liczne problemy do rozwiązania w domu. Wielu wykładowców zechce uzupełnić ten materiał przedsięwzięciami. W tym dodatku zostały zawarte pewne ułatwiające to zadanie wskazówki. Przedstawiono również materiały pomocnicze dostępne w podręczniku wykładowcy. Materiały pomocnicze obejmują trzy rodzaje przedsięwzięć:

- Wyszukiwanie informacji
- Zadania symulacyjne
- Zadania typu lektura-sprawozdanie

C.1. Wyszukiwanie informacji

Skutecznym sposobem ułatwiającym przyswajanie podstawowych koncepcji z programu i nauczenie studentów umiejętności wyszukiwania wiedzy jest wykonywanie przez nich przedsięwzięć polegających na wyszukiwaniu informacji na zadany temat. Przedsięwzięcie takie może obejmować przeszukiwanie literatury, a także wyszukiwanie w sieci WWW określonych produktów, prac prowadzonych w laboratoriach badawczych i działań normalizacyjnych. Przedsięwzięcia mogą być przypisywane zespołom lub – jeśli są niewielkie – pojedynczym osobom. W każdym przypadku najlepiej jest wymagać pewnego rodzaju propozycji przedsięwzięcia na początku semestru, co dałoby wykładowcy czas na ocenę tematu i przewidywanego wkładu pracy. Studenci powinni otrzymywać następujące materiały:

- Format propozycji
- Format sprawozdania końcowego
- Harmonogram z terminami pośrednimi i z terminem końcowym
- Wykaz możliwych tematów przedsięwzięć

Student może wybrać jeden spośród wymienionych tematów lub zaproponować własne, porównywalne przedsięwzięcie. Podręcznik wykładowcy zawiera sugerowane formaty propozycji i sprawozdania końcowego, a także wykaz możliwych tematów.

C.2. Zadania symulacyjne

Doskonałym sposobem poznania wewnętrznego funkcjonowania procesora oraz studiowania i opanowania pewnych kompromisów projektowych i implikacji w odniesieniu do wydajności jest symulowanie podstawowych składników procesora. Dwoma narzędziami, które są przydatne do tego celu, są SimpleScalar i SMPCache.

W porównaniu z rzeczywistymi implementacjami sprzętowymi symulacja ma dwie zalety zarówno w zastosowaniach badawczych, jak i edukacyjnych:

- Za pomocą symulacji łatwo jest modyfikować różne elementy organizacji i zmieniać parametry wydajności różnych składników, po czym analizować wyniki takich modyfikacji.
- Symulacja umożliwia gromadzenie szczegółowych danych statystycznych dotyczących wydajności, które mogą posłużyć do lepszego zrozumienia kompromisów towarzyszących poszczególnym rozwiązaniom.

SimpleScalar

SimpleScalar [BURG97, MANJ01a, MANJ01b] to zbiór narzędzi, które mogą być używane do symulowania rzeczywistych programów w szerokim zakresie nowoczesnych procesorów i systemów. Zbiór ten obejmuje kompilator, asembler, program łączący oraz narzędzia symulacji i wizualizacji. SimpleScalar zawiera symulatory procesorów, począwszy od krańcowo szybkich symulatorów funkcjonalnych, aż do szczegółowego symulatora procesora superskalarnego opartego na wydawaniu rozkazów w zmienionej kolejności, który wspiera nieblokujące pamięci podręczne i wykonywanie spekulatywne. Architektura listy rozkazów i parametry organizacji mogą być modyfikowane, co umożliwia prowadzenie różnorodnych eksperymentów.

Podręcznik wykładowcy związany z tą książką zawiera zwięzłe wprowadzenie do programu SimpleScalar przeznaczone dla studentów, wraz ze wskazówkami dotyczącymi pobierania i uruchamiania tego programu. Znajdują się w nim również sugerowane przedsięwzięcia, które mogą być zadawane studentom.

SimpleScalar to przenośny pakiet oprogramowania, działający na większości platform UNIX. Może on być pobrany z witryny WWW SimpleScalar. Do zastosowań niekomercyjnych jest dostępny bezpłatnie.

SMPCache

SMPCache to symulator śledzący służący do analizowania i nauczania systemów pamięci podręcznych w wieloprocesorach symetrycznych [RODR01]. Symulacja jest oparta na modelu zbudowanym zgodnie z podstawowymi zasadami architektury tych systemów. Symulator ma w pełni graficzny i przyjazny interfejs. Oto niektóre spośród parametrów, które mogą być badane za pomocą tego symulatora: lokalność programów, wpływ liczby procesorów, protokołów spójności pamięci podręcznych,

sposobów prowadzenia arbitrażu magistrali, odwzorowania, metod zastępowania, rozmiarów pamięci podręcznych (bloków wewnątrz tych pamięci), liczby sekcji pamięci podręcznych (w przypadku pamięci sekcyjno-skojarzeniowych) i liczby słów w bloku (rozmiaru bloku pamięci).

Podręcznik wykładowcy związany z tą książką zawiera zwięzłe wprowadzenie do programu SMPCache przeznaczone dla studentów, wraz ze wskazówkami dotyczącymi pobierania i uruchamiania tego programu. Znajdują się w nim również sugerowane przedsięwzięcia, które mogą być zadawane studentom.

SMPCache to przenośny pakiet oprogramowania, działający w systemach PC z Windows. Może on być pobrany z witryny WWW SMPCache. Do zastosowań niekomercyjnych jest dostępny bezpłatnie.

C.3. Zadania typu lektura-sprawozdanie

Innym doskonałym sposobem utrwalania koncepcji przekazywanych w ramach wykładu i kształtowania u studentów umiejętności wyszukiwania informacji jest zobowiązywanie tych studentów do przeczytania i przeanalizowania określonych artykułów z literatury.

W podręczniku wykładowcy znajduje się sugerowany wykaz artykułów (po jednym lub po dwa na rozdział). Wszystkie są łatwo dostępne poprzez Internet lub w dowolnej, dobrej bibliotece technicznej na uczelni. Podręcznik zawiera również sugerowany sposób formułowania zadań tego rodzaju.

Słownik

Niektóre terminy w tym słowniku pochodzą z *American National Directory for Information Systems* (1996). Zostały one zaznaczone gwiazdką.

Adres bazowy* (*base address*) Wartość numeryczna używana jako odniesienie przy obliczaniu adresów podczas wykonywania programu komputerowego.

Adres bezpośredni* (*direct address*) Adres określający miejsce przechowywania elementu danych, który ma być traktowany jako argument. Synonim *adresu jednopoziomowego*.

Adres bezwzględny* (*absolute address*) Adres w języku komputerowym identyfikujący miejsce przechowywania lub urządzenie bez użycia jakiegokolwiek odniesienia pośredniego.

Adres indeksowany* (*indexed address*) Adres, który jest modyfikowany przez zawartość rejestru modyfikacji przed lub podczas wykonywania rozkazu komputerowego.

Adres natychmiastowy (*immediate address*) Część adresowa zawierająca wartość argumentu, a nie jego adres. Synonim *adresu zeropoziomowego*.

Adres pośredni* (*indirect address*) Adres miejsca przechowywania, które zawiera adres.

Akumulator (*accumulator*) Nazwa rejestru procesora w jednoadresowym formacie rozkazu. Akumulator (AC) zawiera domyślnie jeden lub dwa argumenty rozkazu.

Arbitraż magistrali (*bus arbitration*) Proces określania, która spośród rywalizujących ze sobą jednostek nadrzędnych uzyska zezwolenie na dostęp do magistrali.

Argument* (*operand*) Obiekt, na którym jest wykonywana operacja.

ASCII Kod 7-bitowy używany do reprezentowania drukowalnych znaków numerycznych, alfabetycznych i specjalnych. Obejmuje również kody znaków sterujących, które nie są drukowane ani wyświetlane, lecz określają pewną funkcję sterowania.

Bajt (*byte*) Osiem bitów. Określany również jako oktet.

Bardzo długie słowo rozkazu (VLIW) (*very long instruction word*) Odnosi się do stosowania rozkazów zawierających wiele operacji. W rezultacie w jednym słowie jest zawartych wiele rozkazów. Zwykle VLIW są tworzone przez kompilator umieszczający w tym samym słowie operacje, które mogą być wykonywane równolegle.

Bit* W czystym binarnym systemie liczenia, jedna z cyfr 0 i 1.

Bit parzystości* (*parity bit*) Cyfra binarna dołączona do grupy cyfr binarnych w ten sposób, że suma wszystkich cyfr jest zawsze nieparzysta (bit nieparzystości) lub parzysta (bit parzystości).

Blok sterowania procesami (*process control block*) Reprezentacja procesu w systemie operacyjnym. Jest to struktura danych zawierająca informację o właściwościach i stanie procesu.

Blok stronicowy* (*page frame*) Obszar pamięci głównej używany do przechowywania strony.

Błąd strony (*page fault*) Występuje, gdy strona zawierająca słowo, do którego nastąpiło odniesienie, nie znajduje się w pamięci głównej. Powoduje to przerwanie i wymaga, żeby system operacyjny dostarczył potrzebną stronę.

Bramka (*gate*) Układ elektroniczny, którego sygnał wyjściowy jest prostą funkcją Boole'a jego sygnałów wejściowych.

Bufor* (*buffer*) Pamięć używana do skompensowania różnicy szybkości przepływu danych lub czasu następowania zdarzeń podczas przesyłania danych z jednego przyrządu do drugiego.

CD-ROM Pamięć stała na dysku kompaktowym. Niewymazywalny dysk używany do przechowywania danych komputerowych. Standardowe dyski mają średnicę 12 cm i pojemność ponad 650 MB.

Cykl adresowania pośredniego (*indirect cycle*) Część cyklu rozkazu, podczas której procesor wykonuje operację dostępu do pamięci w celu przekształcenia adresu pośredniego na bezpośredni.

Cykl pobierania (*fetch cycle*) Część cyklu rozkazu, podczas której procesor pobiera z pamięci rozkaz przeznaczony do wykonywania.

Cykl przerwania (*interrupt cycle*) Część cyklu rozkazu, podczas której procesor sprawdza, czy nie wystąpiło przerwanie. Jeśli w stanie zawieszenia znajduje się przerwanie dozwolone, to zachowuje on stan bieżący programu i wznawia przetwarzanie po zrealizowaniu programu obsługi przerwania.

Cykl rozkazu (*instruction cycle*) Przetwarzanie prowadzone przez procesor w celu wykonania pojedynczego rozkazu.

Cykl wykonywania (*execute cycle*) Część cyklu rozkazu, podczas której procesor wykonuje operację określoną przez kod operacji.

Czas cyklu pamięci (*memory cycle time*) Odwrotność częstości, z jaką mogą następować operacje dostępu do pamięci. Jest to minimalny czas między odpowiedzią na jedno zgłoszenie dostępu (odczytu lub zapisu) a odpowiedzią na następne zgłoszenie dostępu.

Czas cyklu procesora (*processor cycle time*) Czas potrzebny do wykonania najkrótszej, ściśle określonej mikrooperacji procesora. Jest to podstawowa jednostka do mierzenia czasu trwania wszystkich działań procesora. Synonim *czasu cyklu maszynowego*.

Dekoder* (*decoder*) Przyrząd mający pewną liczbę linii wejściowych, z których dowolna liczba może przenosić sygnały, oraz pewną liczbę linii wyjściowych, z których nie więcej niż jedna może przenosić sygnał, przy czym istnieje jednoznaczne przyporządkowanie między wyjściami a kombinacjami sygnałów wejściowych.

Dostęp bezpośredni* (*direct access*) Możliwość uzyskania danych z urządzenia pamięciowego lub wprowadzenia danych do tego urządzenia w sekwencji niezależnej od względnego położenia tych danych, za pomocą adresu wskazującego fizyczną lokację danych.

Dostęp bezpośredni do pamięci (DMA) (*direct memory access*) Postać wejścia-wyjścia, w której specjalny moduł, nazywany modułem DMA, steruje wymianą danych między pamięcią główną a modułem wejścia-wyjścia. Procesor wysyła zapotrzebowanie na przesłanie bloku danych do modułu DMA. Wygenerowanie przerwania przez procesor następuje dopiero po zakończeniu transferu całego bloku.

Duży komputer (*mainframe*) Określenie odnoszące się początkowo do szafy zawierającej jednostkę centralną w dużym komputerze pracującym w trybie wsadowym. Po pojawieniu się minikomputerów na początku lat siedemdziesiątych tradycyjne, duże komputery były określane jak *mainframes*. Typowymi właściwościami dużych komputerów są: obsługiwanie wielkich baz danych, rozbudowane urządzenia wejścia-wyjścia oraz zastosowanie w scentralizowanych zakładach przetwarzania danych.

Dysk kompaktowy (CD) (*compact disk*) Niewymazywalny dysk przechowujący informację audio w postaci cyfrowej.

Dysk magnetyczny* (*magnetic disk*) Płaska, okrągła płytka pokryta warstwą magnetyczną, na której powierzchni mogą być przechowywane dane wprowadzone metodą zapisu magnetycznego.

Dyskietka* (*diskette*) Elastyczny dysk magnetyczny zamknięty w obudowie ochronnej. Synonim *dysku elastycznego*.

Emulacja* (*emulation*) Imitowanie całości lub części jednego systemu przez inny system o charakterze przede wszystkim sprzętowym, dzięki czemu system imitujący akceptuje takie same dane, wykonuje takie same programy i uzyskuje takie same wyniki co system imitowany.

Format rozkazu (*instruction format*) Struktura rozkazu komputerowego jako sekwencji bitów. Format dzieli rozkaz na pola odpowiadające elementom składowym rozkazu (np. kod operacji, argumenty).

G Przedrostek oznaczający miliard.

Indeksowanie (*indexing*) Metoda modyfikacji adresu za pomocą rejestrów indeksowych.

Indeksowanie automatyczne (*autoindexing*) Forma adresowania indeksowego, w której zawartość rejestru indeksowego jest automatycznie inkrementowana lub dekrementowana za każdym odniesieniem do pamięci.

Izolowane wejście-wyjście (*isolated I/O*) Metoda adresowania modułów wejścia-wyjścia i urządzeń zewnętrznych. Przestrzeń adresowa wejścia-wyjścia jest traktowana oddzielnie od przestrzeni adresowej pamięci głównej. Muszą być używane specjalne rozkazy maszynowe wejścia-wyjścia. Porównaj *wejście-wyjście odwzorowane w pamięci*.

Jądro (*nucleus*) Część systemu operacyjnego zawierająca jego podstawowe i najczęściej używane funkcje. Często jądro rezyduje w pamięci głównej na stałe.

Jednostka arytmetyczno-logiczna (ALU)* (*arithmetic and logic unit*) Część komputera, która wykonuje operacje arytmetyczne, logiczne i pokrewne.

Jednostka centralna (CPU) (*central processing unit*) Część komputera, która pobiera i wykonuje rozkazy. Składa się z jednostki arytmetyczno-logicznej, jednostki sterującej i rejestrów. Często jest określana po prostu jako procesor.

Jednostka nadrzędna magistrali (*bus mater*) Przyrząd przyłączony do magistrali, który może inicjować i kontrolować komunikację na magistrali.

Jednostka sterująca (*control unit*) Część procesora, która steruje jego operacjami, w tym operacjami ALU, przenoszeniem danych wewnątrz procesora oraz wymianą danych i sygnałów sterujących poprzez interfejsy zewnętrzne (np. magistralę systemową).

Język asemblerowy* (*assembly language*) Język komputerowy, którego rozkazy zwykle odpowiadają rozkazom komputera i który może zawierać udogodnienia w postaci makrorozkazów. Synonim *języka zależnego od komputera*.

Język mikroprogramowania (*microprogramming language*) Lista rozkazów służąca do tworzenia mikroprogramów.

K Przedrostek oznaczający $2^{10} = 1024$. Wobec tego 2 Kbit = 2048 bitów.

Kanał bajtowo-multiplekserowy* (*byte multiplexer channel*) Kanał multiplekserowy, który przeplata przekazywane bajty danych. Patrz również *kanał multiplekserowy z blokową transmisją danych*. Rozwiązanie przeciwstawne – *kanał wybiórczy*.

Kanał multiplekserowy (*multiplexer channel*) Kanał przeznaczony do jednoczesnej pracy z wieloma urządzeniami wejścia-wyjścia. Kilka urządzeń wejścia-wyjścia może przekazywać rekordy w tym samym czasie dzięki przeplatanej transmisji elementów danych. Patrz również *kanał bajtowo-multiplekserowy* i *kanał multiplekserowy z blokową transmisją danych*.

Kanał multiplekserowy z blokową transmisją danych (*block multiplexer channel*) Kanał multiplekserowy, który przeplata przekazywane bloki danych. Patrz również *kanał bajtowo-multiplekserowy*. Rozwiązanie przeciwstawne – *kanał wybiórczy*.

Kanał wejścia-wyjścia (*I/O channel*) Stosunkowo złożony moduł wejścia-wyjścia, który uwalnia procesor od szczegółów operacji wejścia-wyjścia. Kanał wejścia-wyjścia wykona sekwencję rozkazów wejścia-wyjścia z pamięci głównej bez potrzeby angażowania procesora.

Kanał wybiórczy (*selector channel*) Kanał wejścia-wyjścia przeznaczony do współpracy z tylko jednym urządzeniem wejścia-wyjścia w określonym czasie. Gdy urządzenie wejścia-wyjścia zostało wybrane, kompletny rekord jest przekazywany bajt po bajcie. Rozwiązanie przeciwstawne – *kanał multiplekserowy z blokową transmisją danych* oraz *kanał multiplekserowy*.

Klaster (*cluster*) Grupa połączonych ze sobą kompletnych komputerów, współpracujących jako jednolity system przetwarzania, stwarzający wrażenie, że jest jednym komputerem. Wyrażenie *kompletny komputer* oznacza tutaj system zdolny do samodzielnej pracy poza klastrem.

Kod detekcyjny* (*error-detecting code*) Kod, w którym każdy znak lub sygnał spełniają określone reguły budowy, dzięki czemu odchylenia od tych reguł wskazują na obecność błędu.

Kod korekcyjny* (*error correcting code*) Kod, w którym każdy znak lub sygnał spełniają określone reguły budowy (dzięki czemu odchylenia od tych reguł wskazują na obecność błędu) i w którym niektóre lub wszystkie wykryte błędy mogą być automatycznie poprawiane.

Kod operacji* (*operation code – opcode*) Kod używany do reprezentowania operacji komputera.

Kod warunkowy (*condition code*) Kod odzwierciedlający wynik poprzedniej operacji (np. arytmetycznej). Procesor może obejmować jeden lub wiele kodów warunkowych, które mogą być przechowywane oddzielnie wewnątrz procesora lub jako część większego rejestru sterowania. Znany także jako *znaczniki stanu*.

Komunikacja danych (*data communication*) Transfer danych między urządzeniami. Na ogół wyłącza się z tego zakresu urządzenia wejścia-wyjścia.

Licznik rozkazów (*program counter*) Rejestr adresów rozkazów.

Lista rozkazów* (*computer instruction set*) Kompletny zbiór operatorów rozkazów komputera wraz z opisem rodzajów znaczeń, które mogą być przypisane ich argumentom. Synonim *listy rozkazów maszynowych*.

Lokalność odniesienia (*locality of reference*) Skłonność procesora do powtarzalnego sięgania do tego samego zbioru lokacji pamięci w krótkim okresie.

M Przedrostek oznaczający 2^{20} = 1 048 576. Wobec tego 2 Mbit = 2×2^{20} bitów.

Magistrala (*bus*) Wspólna ścieżka komunikacji składająca się z jednej linii lub ze zbioru linii. W niektórych systemach komputerowych procesor, pamięć oraz urządzenia wejścia-wyjścia są połączone za pomocą wspólnej magistrali. Ponieważ linie są używane wspólnie przez wszystkie zespoły, tylko jeden z nich może transmitować w określonej chwili.

Magistrala systemowa (*system bus*) Magistrala służąca do łączenia głównych zespołów komputera (procesora, pamięci, wejścia-wyjścia).

Mikrokomputer* (*microcomputer*) System komputerowy, którego procesor jest mikroprocesorem. Podstawowy mikrokomputer zawiera mikroprocesor, pamięć i urządzenia wejścia-wyjścia, które mogą, lecz nie muszą znajdować się w tym samym mikroukładzie.

Mikrooperacja (*microoperation*) Elementarna operacja procesora, wykonywana podczas jednego impulsu zegara.

Mikroprocesor* (*microprocessor*) Procesor, którego elementy zostały zminiaturyzowane w postaci jednego lub kilku układów scalonych.

Mikroprogram* (*microprogram*) Sekwencja mikrorozkazów znajdująca się w specjalnej pamięci, z której mogą być dynamicznie pobierane w celu wykonywania różnych funkcji.

Mikroprogramowana jednostka centralna (*microprogrammed CPU*) Procesor, którego jednostka sterująca została wdrożona przy użyciu mikroprogramowania.

Mikrorozkaz* (*microinstruction*) Rozkaz sterujący przepływem danych i szeregowaniem w procesorze na poziomie bardziej podstawowym niż rozkazy maszynowe. Pojedyncze rozkazy maszynowe i być może inne funkcje mogą być realizowane za pomocą mikroprogramów.

Moduł wejścia-wyjścia (*I/O module*) Jeden z głównych rodzajów zespołów komputera. Jest odpowiedzialny za sterowanie jednym lub wieloma urządzeniami zewnętrznymi (peryferyjnymi) oraz za wymianę danych między tymi urządzeniami a pamięcią główną i (lub) rejestrami procesora.

Multiplekser (*multiplexer*) Układ kombinacyjny, który łączy wiele wejść z jednym wyjściem. W dowolnej chwili tylko jedno z wejść jest wybrane jako połączone z wyjściem.

Operator binarny* (*binary operator*) Operator, który reprezentuje operację na dwóch i tylko dwóch argumentach.

Operator jednoargumentowy* (*unary operator*) Operator, który reprezentuje operację na jednym i tylko jednym argumencie.

Oprogramowanie układowe* (*firmware*) Mikroprogram przechowywany w pamięci stałej.

Ortogonalność (*orthogonality*) Zasada, zgodnie z którą dwie zmienne lub wymiary są od siebie wzajemnie niezależne. W kontekście listy rozkazów termin ten jest używany do wskazania, że pozostałe składniki rozkazu (tryb adresowania, liczba argumentów, długość argumentu) są niezależne od kodu operacji (nie są przez ten kod wyznaczone).

Pakiet dysków* (*disk pack*) Zespół dysków magnetycznych, który może być usunięty jako całość z napędu, łącznie z pojemnikiem, od którego zespół musi być oddzielony podczas pracy.

Pamięć dynamiczna RAM (*dynamic RAM*) Pamięć RAM, której komórki są zrealizowane w postaci kondensatorów. Dynamiczne pamięci RAM stopniowo tracą swoją zawartość, czemu zapobiega okresowe odświeżanie.

Pamięć główna* (*main memory*) Pamięć adresowalna przez program, z której rozkazy i inne dane mogą być ładowane bezpośrednio do rejestrów w celu dalszego wykonywania lub przetwarzania.

Pamięć nieulotna (*nonvolatile memory*) Pamięć, której zawartość jest stabilna i nie wymaga stałego zasilania.

Pamięć o dostępie swobodnym (RAM) (*random-access memory*) Pamięć, w której każda adresowalna lokacja ma unikatowy sposób adresowania. Czas dostępu do danej lokacji jest niezależny od sekwencji poprzednich dostępów.

Pamięć podręczna* (*cache memory*) Specjalna pamięć buforowa, mniejsza i szybsza niż pamięć główna, używana do przechowywania kopii tych rozkazów i danych z pamięci głównej, które najprawdopodobniej będą potrzebne procesorowi jako następne i które zostały automatycznie uzyskane z pamięci głównej.

Pamięć pomocnicza (*secondary memory*) Pamięć umieszczona poza samym systemem komputerowym, w tym dyskowa i taśmowa.

Pamięć skojarzeniowa* (*associative memory*) Pamięć, w której lokacje są identyfikowane przez ich zawartość lub częściowo przez ich zawartość, nie zaś przez ich nazwy czy położenie.

Pamięć tylko do odczytu (stała) (ROM) (*read-only memory*) Pamięć półprzewodnikowa, której zawartość nie może być zmieniana bez zniszczenia tej pamięci. Pamięć niewymazywalna.

Pamięć sterująca (*control storage*) Część pamięci zawierająca mikroprogram.

Pamięć ulotna (*volatile memory*) Pamięć, w przypadku której w celu zachowania zawartości jest wymagane ciągłe zasilanie elektryczne. Po wyłączeniu zasilania przechowywana informacja jest stracona.

Pamięć wirtualna* (*virtual storage*) Przestrzeń pamięci, która może być traktowana przez użytkownika systemu komputerowego jako adresowalna pamięć główna i w której adresy wirtualne są odwzorowane w zbiorze adresów rzeczywistych. Rozmiar pamięci wirtualnej jest ograniczony przez schemat adresowania systemu komputerowego oraz przez wielkość dostępnej pamięci pomocniczej, nie zaś przez rzeczywistą liczbę lokacji w pamięci głównej.

Paskowanie dysków (*disk stripping*) Rodzaj organizacji danych w zespole dysków, polegający na tym, że logicznie sąsiadujące bloki danych (lub paski danych) są rozlokowane cyklicznie na kolejnych dyskach zespołu. Zbiór logicznie sąsiadujących pasków danych, odwzorowany w postaci dokładnie jednego paska na każdym dysku w zespole, jest określany jako pasek.

Podstawa naukowego systemu liczenia* (*base*) W naukowym systemie liczenia jest to liczba, która jest podnoszona do potęgi określonej przez wykładnik, a następnie mnożona przez mantysę w celu otrzymania rzeczywistej reprezentowanej liczby (np. liczba 10 w wyrażeniu $2{,}7 \times 10^2 = 270$).

Podzespoły na ciele stałym* (*solid-state component*) Podzespoły, których działanie zależy od sterowania zjawiskami elektrycznymi lub magnetycznymi w ciałach stałych (np. tranzystor, dioda półprzewodnikowa, rdzeń ferrytowy).

Potok (*pipeline*) Organizacja procesora, w której procesor składa się z pewnej liczby stopni (etapów), co umożliwia równoczesne wykonywanie wielu rozkazów.

Półprzewodnik (*semiconductor*) Stała substancja krystaliczna, taka jak krzem lub german, której przewodnictwo elektryczne jest pośrednie między izolatorami a przewodnikami*. Półprzewodników używa się do wytwarzania tranzystorów i innych podzespołów na ciele stałym.

Predykatywne wykonywanie rozkazów (*predicated execution*) Mechanizm warunkowego wykonywania pojedynczych rozkazów. Umożliwia spekulatywne wykonywanie obydwu gałęzi rozkazu rozgałęzienia i zachowywanie wyników rozgałęzienia do czasu, gdy ono nastąpi.

Proces (*process*) Wykonywany program. Proces jest sterowany i szeregowany przez system operacyjny.

---

* Bardziej istotną cechą półprzewodników jest wrażliwość ich przewodnictwa elektrycznego na pole elektryczne, magnetyczne, światło i temperaturę (*przyp. tłum.*).

Procesor* (*processor*) Jednostka funkcjonalna komputera, która interpretuje i wykonuje rozkazy. Procesor składa się przynajmniej z jednostki sterującej i jednostki arytmetycznej.

Procesor superpotokowy (*superpipeline processor*) Rozwiązanie procesora, w którym potok rozkazów składa się z wielu bardzo małych etapów, dzięki czemu w jednym cyklu zegara jest możliwe wykonywanie więcej niż jednego etapu potoku, w wyniku czego w potoku może się jednocześnie znajdować wielka liczba rozkazów.

Procesor superskalarny (*superscalar processor*) Rodzaj procesora, który zawiera wiele potoków rozkazów, dzięki czemu na tym samym etapie potoku można wykonywać więcej niż jeden rozkaz.

Procesor wejścia-wyjścia (*I/O processor*) Moduł wejścia-wyjścia z własnym procesorem, który może wykonywać specjalistyczne rozkazy maszynowe wejścia-wyjścia, a także, w pewnych przypadkach, rozkazy maszynowe o ogólnym przeznaczeniu.

Procesor z przetwarzaniem superpotokowym (*superpipelined processor*) Rodzaj procesora, w którym potok rozkazów składa się z bardzo małych etapów, w wyniku czego więcej niż jeden etap potoku może być wykonany podczas jednego cyklu zegara. Dzięki temu wielka liczba rozkazów może się jednocześnie znajdować w potoku.

Programowalna pamięć stała (PROM) (*programmable read-only memory*) Pamięć półprzewodnikowa, której zawartość może być ustalona jednorazowo. Proces zapisu jest realizowany elektrycznie i może być dokonywany przez użytkownika po wytworzeniu oryginalnego mikroukładu.

Programowalna tablica logiczna (PLA)* (*programmable logic array*) Tablica bramek, których połączenia mogą być programowane w celu realizowania określonej funkcji logicznej.

Programowane wejście-wyjście (*programmed I/O*) Forma wejścia-wyjścia, w której procesor wydaje modułowi wejścia-wyjścia rozkaz wejścia-wyjścia, po czym musi czekać na zakończenie operacji, zanim będzie kontynuował działanie.

Protokół spójności pamięci podręcznej (*cache coherence protocol*) Mechanizm utrzymywania ważności danych w wielu pamięciach podręcznych, tak aby każdy dostęp do danych prowadził do uzyskania najnowszej wersji zawartości słowa pamięci głównej.

Przerwanie* (*interrupt*) Zawieszenie procesu, takiego jak wykonywanie programu komputerowego, spowodowane przez zdarzenie zewnętrzne w stosunku do tego procesu i wykonywane w taki sposób, że proces może być wznowiony.

Przerwanie dozwolone (*enabled interrupt*) Warunek, tworzony zwykle przez procesor, podczas którego odpowiada on na sygnały zapotrzebowania na przerwanie określonej klasy.

Przerwanie zablokowane (*disabled interrupt*) Warunek, tworzony zwykle przez procesor, podczas którego ignoruje on sygnały zapotrzebowania na przerwanie określonej klasy.

Przerzutnik* (*flip-flop*) Układ lub przyrząd zawierający elementy aktywne. Może on przyjmować w danym czasie jeden z dwóch stanów stabilnych. Synonim *układu dwustabilnego*.

Przestrzeń adresowa (*address space*) Zakres adresów (pamięci, wejścia-wyjścia), do których mogą następować odniesienia.

Przetwarzanie jednoprocesorowe (*uniprocessing*) Sekwencyjne wykonywanie rozkazów przez procesor lub niezależne używanie procesora w systemie wieloprocesorowym.

Przetwarzanie o wysokiej wydajności (HPC) (*high-performance computing*) Obszar badań obejmujący superkomputery i przeznaczone dla nich oprogramowanie. Nacisk jest kładziony na zastosowania naukowe, w których występuje intensywne użycie obliczeń wektorowych i matrycowych oraz algorytmy równoległe.

Przewidywanie rozgałęzień (*branch prediction*) Mechanizm używany przez procesor do przewidywania wyniku rozgałęzienia programu przed jego wykonaniem.

Ramka strony (*page frame*) Część pamięci głównej używana do przechowywania strony.

Regulacja asynchroniczna (*asynchronous timing*) Metoda, w której występowanie jednego zdarzenia na magistrali jest zależne od wystąpienia poprzedniego zdarzenia.

Regulacja synchroniczna (*synchronous timing*) Metoda, w której występowanie zdarzeń na magistrali jest wyznaczane przez zegar. Zegar określa jednakowej długości przedziały czasowe, a zdarzenia mogą się rozpoczynać jedynie na początku takiego przedziału.

Rejestr adresowy pamięci (MAR)* (*memory address register*) Rejestr w procesorze zawierający adres tej lokacji w pamięci, do której następuje dostęp.

Rejestr adresu rozkazu* (*instruction address register*) Specjalistyczny rejestr służący do przechowywania adresu następnego rozkazu przewidzianego do wykonania.

Rejestr buforowy pamięci (MBR) (*memory buffer register*) Rejestr zawierający dane odczytane w pamięci lub dane, które mają być zapisane w pamięci.

Rejestr indeksowy* (*index register*) Rejestr, którego zawartość może być użyta do zmodyfikowania adresu argumentu podczas wykonywania rozkazów komputerowych; może być również użyty jako licznik. Rejestr indeksowy może służyć do sterowania wykonywaniem pętli, do sterowania wykorzystaniem tablic, jako przełącznik, do przeglądania tablic lub jako wskaźnik.

Rejestr ogólnego przeznaczenia (roboczy)* (*general-purpose register*) Rejestr, zwykle jawnie adresowalny w ramach zbioru rejestrów, który może być używany do różnych celów, na przykład jako akumulator, jako rejestr indeksowy lub jako specjalny rejestr danych.

Rejestr rozkazu* (*instruction register*) Rejestr używany do przechowywania rozkazu przeznaczonego do interpretacji.

Rejestry (*registers*) Szybkie pamięci wewnętrzne procesora. Niektóre rejestry są widzialne dla użytkownika, to znaczy dostępne dla programisty poprzez listę rozkazów maszynowych. Inne rejestry są używane wyłącznie przez procesor do celów sterowania.

Rejestry sterujące (*control registers*) Rejestry procesora służące do sterowania jego działaniem. Większość z nich jest niewidzialna dla użytkownika.

Rejestry widzialne dla użytkownika (*user-visible registers*) Rejestry procesora, do których może się odnosić programista. Format listy rozkazów umożliwia to, żeby jeden lub więcej rejestrów było przeznaczonych do przechowywania argumentów lub adresów argumentów.

Reprezentacja dopełnienia do dwóch (*twos complement representation*) Używana do reprezentowania całkowitych liczb binarnych. Dodatnia całkowita liczba binarna jest reprezentowana w notacji znak-moduł. Liczba ujemna jest reprezentowana przez dodanie 1 do reprezentacji uzupełnienia do 1 tej samej liczby.

Reprezentacja znak-moduł (*sign-magnitude representation*) Używana do reprezentowania całkowitych liczb binarnych. W słowie N-bitowym lewy bit określa znak (0 = dodatni, 1 = ujemny), a pozostałych $N - 1$ bitów określa wielkość.

Rozkaz komputerowy* (*computer instruction*) Rozkaz, który może być rozpoznany przez procesor komputera, dla którego został zaprojektowany. Synonim *rozkazu maszynowego*.

Skalar* (*scalar*) Wielkość określana za pomocą pojedynczej wartości.

Skok bezwarunkowy* (*unconditional jump*) Skok, który następuje przy każdym wykonywaniu określonego rozkazu.

Skok warunkowy* (*conditional jump*) Skok, który następuje tylko wtedy, kiedy jest wykonywany określony rozkaz i gdy jednocześnie są spełnione określone warunki; w odróżnieniu od *skoku bezwarunkowego*.

Słowo stanu programu (PSW) (*program status word*) Obszar w pamięci używany do wskazywania kolejności wykonywania rozkazów oraz do przechowywania i wskazywania stanu systemu komputerowego. Synonim *słowa stanu procesora*.

Statyczna pamięć o dostępie swobodnym (*static RAM*) Pamięć RAM, której komórki zostały zrealizowane za pomocą przerzutników. Statyczna pamięć RAM przechowuje dane tak długo, jak długo jest zasilana; nie jest wymagane okresowe odświeżanie.

Sterownik wejścia-wyjścia (*I/O controller*) Stosunkowo prosty moduł wejścia-wyjścia, który wymaga szczegółowego sterowania ze strony procesora lub kanału wejścia-wyjścia. Synonim *sterownika urządzenia*.

Stos* (*stack*) Lista zbudowana i utrzymywana w ten sposób, że następna pozycja przewidziana do wydobycia jest tą pozycją, która ostatnio została na tej liście umieszczona (ostatnia na wejściu, pierwsza na wyjściu – LIFO).

Strona* (*page*) W systemie pamięci wirtualnej blok o ustalonej długości, który ma adres wirtualny i który jest przenoszony jako całość między pamięcią rzeczywistą a pamięcią pomocniczą.

Stronicowanie na żądanie* (*demand paging*) Transfer strony z pamięci pomocniczej do rzeczywistej w czasie, gdy jest potrzebna.

System operacyjny* (*operating system*) Oprogramowanie sterujące wykonywaniem programów i realizujące takie usługi, jak rozdzielanie zasobów, szeregowanie, sterowanie wejściem-wyjściem i zarządzanie danymi.

System reprezentacji liczb stałopozycyjnych* (*fixed-point representation system*) System liczenia o stałej podstawie, w którym miejsce przecinka jest domyślnie ustalone w ciągu cyfr za pomocą uzgodnionej konwencji.

Szyna adresowa (*address bus*) Część magistrali systemowej używana do przesyłania adresu. Adres określa zwykle lokację w pamięci głównej lub urządzenie wejścia--wyjścia.

Szyna danych (*data bus*) Część magistrali systemowej używana do przesyłania danych.

Szyna sterująca (*control bus*) Część magistrali systemowej używana do przesyłania sygnałów sterujących.

Tablica niezależnych dysków twardych (RAID) (*redundant array of independent disks*) Tablica dysków, w której część fizycznej pojemności pamięci jest używana do przechowywania nadmiarowej (redundantnej) informacji o danych użytkownika zapisanych w pozostałej części pamięci. Informacja nadmiarowa umożliwia odtworzenie danych użytkownika, gdy jeden z dysków wchodzących w skład tablicy lub ścieżka dostępu do niego są uszkodzone.

Tablica prawdy* (*truth table*) Tablica opisująca funkcję logiczną przez wymienienie wszystkich możliwych wartości wejściowych i wskazanie wartości wyjściowej odpowiadającej każdej z tych kombinacji.

Taśma magnetyczna* (*magnetic tape*) Taśma z magnetyczną warstwą powierzchniową, na której można dokonywać magnetycznego zapisu danych.

Transmisja danych (*data communication*) Przenoszenie danych między urządzeniami. Termin ten zwykle obejmuje wejście-wyjście.

Układ kombinacyjny* (*combinational circuit* lub *combinatorial circuit*) Przyrząd logiczny, którego wartości wyjściowe w dowolnej chwili zależą wyłącznie od wartości wejściowych w tym samym czasie. Układ kombinacyjny jest szczególnym przypadkiem układu sekwencyjnego, który nie umożliwia przechowywania.

Układ łańcuchowy* (*daisy chain*) Metoda łączenia przyrządów służąca do określania priorytetu przerwań, polegająca na szeregowym połączeniu źródeł przerwań.

Układ scalony (IC) (*integrated circuit*) Niewielki kawałek ciała stałego, takiego jak krzem, na którym wytworzono zbiór elementów elektronicznych i ich połączeń.

Układ sekwencyjny (*sequential circuit*) Cyfrowy układ logiczny, którego stan wyjścia zależy od bieżącego stanu wejścia oraz od stanu samego układu. Układy sekwencyjne mają więc atrybut pamięci.

Układ wejścia-wyjścia sterowany przerwaniami (*interrupt-driven I/O*) Forma wejścia-wyjścia. Procesor wydaje rozkaz wejścia-wyjścia i kontynuuje wykonywanie następnych rozkazów. Wykonywanie to ulega przerwaniu przez moduł wejścia--wyjścia, gdy ten ostatni zakończył pracę.

Urządzenie peryferyjne (*peripheral equipment* (IBM)) Jakiekolwiek urządzenie w systemie komputerowym, które zapewnia komunikację procesora z otoczeniem, zewnętrzne w stosunku do tego procesora.

Uzupełnienie do jedynki (*ones complement representation*) Używane do reprezentowania całkowitych liczb binarnych. Dodatnia liczba całkowita jest reprezentowana w postaci znak-moduł. Ujemna liczba całkowita jest reprezentowana przez odwrócenie każdego bitu w reprezentacji dodatniej liczby całkowitej o tej samej wielkości.

Wejście-wyjście (*input-output – I/O*) Odnosi się do wejścia albo do wyjścia, albo do obu tych pojęć. Oznacza przenoszenie danych między komputerem a bezpośrednio dołączonym urządzeniem peryferyjnym.

Wejście-wyjście odwzorowane w pamięci (*memory-mapped I/O*) Metoda adresowania modułów wejścia-wyjścia i urządzeń zewnętrznych. Zarówno w stosunku do pamięci głównej, jak i do wejścia-wyjścia jest używana jedna przestrzeń adresowa. W odniesieniu do odczytu/zapisu w pamięci oraz w urządzeniach wejścia-wyjścia są stosowane takie same rozkazy maszynowe.

Wektor* (*vector*) Wielkość określana zwykle za pomocą uporządkowanego zbioru skalarów.

Wieloprocesor lub **multiprocesor** (*multiprocessor*) Komputer z dwoma lub z większą liczbą procesorów o wspólnym dostępie do pamięci głównej.

Wieloprocesor z niejednorodnym dostępem do pamięci (NUMA) (*nonuniform memory access multiprocessor*) Wieloprocesor o wspólnej pamięci głównej, w którym czas dostępu danego procesora do słowa w pamięci zależy od miejsca tego słowa.

Wieloprogramowanie* (*multiprogramming*) Tryb pracy polegający na przemiennym wykonywaniu dwóch lub większej liczby programów komputerowych przez jeden procesor.

Wieloprzetwarzanie symetryczne (SMP) (*symmetric multiprocessing*) Forma wieloprzetwarzania umożliwiająca systemowi operacyjnemu jednoczesne realizowanie programów w dowolnym dostępnym procesorze lub w kilku dostępnych procesorach.

Wielozadaniowość* (*multitasking*) Tryb pracy umożliwiający jednoczesne lub przeplatane wykonywanie dwóch lub większej liczby zadań komputerowych. To samo co wieloprogramowanie (w innej terminologii).

Wiersz pamięci podręcznej (*cache line*) Blok danych zaopatrzony w znacznik pamięci podręcznej, stanowiący jednostkę transferu między pamięcią podręczną a główną.

Wymazywalny dysk optyczny (*erasable optical disk*) Dysk, w którym zastosowano technologię optyczną, który jednak może być z łatwością wymazany i zapisany ponownie. Używane są dyski o średnicach 3,25 i 5,25 cala. Typowa pojemność dysku wynosi 650 MB.

Zmienna globalna (*global variable*) Zmienna zdefiniowana w jednej części programu komputerowego i używana przez przynajmniej jedną inną część tego programu.

Zmienna lokalna (*local variable*) Zmienna zdefiniowana i używana tylko w jednej, określonej części programu komputerowego.

Zmiennopozycyjny system reprezentacji (*floating-point representation system*) System liczenia, w którym liczba rzeczywista jest reprezentowana przez parę liczb. Pierwsza z nich jest liczbą rzeczywistą będącą iloczynem części stałopozycyjnej, druga zaś jest wartością uzyskaną przez podniesienie domyślnej podstawy do potęgi określonej przez wykładnik.

Literatura

Skróty

ACM Association for Computing Machinery
IEEE Institute of Electrical and Electronics Engineers

ABBO00 Abbot D.: *PCI Bus Demystified.* Eagle Rock, VA: LLH Technology Publishing, 2000.

ACOS86 Acosta R., Kjelstrup J., Torng H.: „An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors". *IEEE Transactions on Computers*, September 1986.

ADAM91 Adamek J.: *Foundations of Coding*. New York, Wiley, 1991.

AGAR89 Agarwal A.: *Analysis of Cache Performance for Operating Systems and Multiprogramming*. Boston, Kluwer Academic Publishers, 1989.

AGER87 Agerwala T., Cocke J.: *High Performance Reduced Instruction Set Processors*. Technical report RC12434 (#55845). Yorktown, IBM Thomas J. Watson Research Center, January 1987.

ALEX93 Alexandridis N.: *Design of Microprocessor-Based Systems*. Englewood Cliffs, Prentice Hall, 1993.

ANDE67a Anderson D., Sparacio F., Tomasulo F.: „The IBM System/360 Model 91: Machine Philosophy and Instruction Handling". *IBM Journal of Research and Development*, January 1967.

ANDE67b Anderson S. et al.: „The IBM System/360 Model 91: Floating-Point Execution Unit". *IBM Journal of Research and Development*, January 1967. Przedruk w [SWAR90, Volume 1].

ANDE98 Anderson D.: *FireWire System Architecture*. Reading, Addison-Wesley, 1998.

ATKI96 Atkins M.: „PC Software Performance Tuning". *IEEE Computer*, August 1996.

AZIM92 Azimi M., Prasad B., Bhat K.: „Two Lewel Cache Architectures". *Proceedings COMPCON '92*, February 1992.

BAEN97 Baentsch M. et al.: „Enhancing the Web's Infrastructure: From Caching to Replication". *Internet Computing*, March/April 1997.

BAIL93 Bailey D.: „RISC Microprocessors and Scientific Computing". *Proceedings, Supercomputing '93*, 1993.

BASH81 Bashe C., Bucholtz W., Hawkins G., Ingram J., Rochester N.: „The Architecture of IBM's Early Computers". *IBM Journal of Research and Development*, September 1981.

BASH91 Bashteen A., Lui I., Mullan J.: „A Superpipeline Approach to the MIPS Architecture". *Proceedings, COMPCON Spring 91*, February 1991.

BELL70 Bell C., Cady R., McFarland H., Delagi B., O'Loughlin J., Noonan R.: „A New Architecture for Minicomputers – the DEC PDP-11". *Proceedings, Spring Joint Computer Conference*, 1970.

BELL71a Bell C., Newell A.: *Computer Structures: Readings and Examples*. New York, McGraw-Hill, 1971.

BELL78a Bell C., Mudge J., McNamara J.: *Computer Engineering: a DEC View of Hardware Systems Design*. Bedford, Digital Press, 1978.

BELL78b Bell C., Newell A., Siewiorek D.: „Structural levels of the PDP-8", w [BELL78a].

BELL78c Bell C., Kotok A., Hastings T., Hill R.: „The Evolution of the DEC System-10". *Communications of the ACM*, January 1978.

BENH92 Benham J.: „A Geometric Approach to Presenting Computer Representations of Integers". *SIGCSE Bulletin*, December 1992.

BETK97 Betker M., Fernando J., Whalen S.: „The history of the Microprocessor". *Bell Labs Technical Journal*, Autumn 1997.

BHAR00 Bharandwaj J. et al.: „The Intel IA-64 Compiler Code Generator". *IEEE Micro*, September/October 2000.

BLAA97 Blaauw G., Brooks F.: *Computer Architecture: Concepts and Evolution.* Reading, Addison-Wesley, 1997.

BLAH83 Blahut R.: *Theory and Practice of Error Control Codes*. Reading, Addison-Wesley, 1983.

BOHR98 Bohr M.: „Silicon Trends and Limits for Advanced Microprocessors". *Communications of the ACM*, March 1998.

BRAD91a Bradlee D., Eggers S., Henry R.: „The Effect on RISC Performance of Register Set Size and Structure Versus Code Generation Strategy". *Proceedings, 18th Annual International Symposium on Computer Architecture*, May 1991.

BRAD91b Bradlee D., Eggers S.: Henry R.: „Integrating Register Allocation and Instruction Scheduling for RISCs". *Proceedings, Forth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

BREW97 Brewer E.: „Clustering: Multiply and Conquer". *Data Communications,* July 1997.

BREY00 Brey B.: *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro and Pentium II Processors*. Upper Saddle River, Prentice Hall, 2000.

BURG97 Burger D., Austin T.: „The SimpleScalar Tool Set, Version 2.0". *Computer Architecture News*, June 1997.

BURK46 Burks A., Goldstine H., von Neumann J.: *Preliminary Discussion of the Logical Design of an Electronic Computer Instrument*. Report prepared for U.S. Army Ordnance Dept., 1946, przedrukowany w [BELL71a].

BUYY99a Buyya R.: *High Performance Cluster Computing: Architectures and Systems.* Upper Saddle River, Prentice Hall 1999.

BUYY99b Buyya R.: *High Performance Cluster Computing: Programming and Applications.* Upper Saddle River, Prentice Hall 1999.

CARM00 Carmean D.: „Inside the High-Performance Intel Processor Microarchitecture". *Intel Developer Forum,* Fall 2000, ftp://download.intel.com/design/id/all2000/presentations/pda/pda_s01_cd.pdf.

CART96 Carter J.: *Microprocessor Architecture and Microprogramming.* Upper Saddle River, Prentice Hall, 1996.

CATA94 Catanzaro B.: *Multiprocessor system architectures.* Mountain View, Sunsoft Press, 1994.

CHAI82 Chaitin G.: „Register Allocation and Spilling via Graph Coloring". *Proceedings, SIGPLAN Symposium on Compiler Construction*, June 1982.

CHAS00 Chasin A.: „Predication, Speculation and Modern CPUs". *Dr. Dobb's Journal*, May 2000.

CHEN94 Chen P., Lee E., Gibson G., Katz R.: „RAID: High Performance, Reliable Secondary Storage". *ACM Computing Surveys*, June 1994.

CHOW86 Chow F., Himmelstein M., Killian E., Weber L.: „Engineering a RISC compiler system". *Proceedings, COMPCON Spring '86*, March 1986.

CHOW87 Chow F., Correll S., Himmelstein M., Killian E., Weber L.: „How Many Addressing Modes Are Enough?" *Proceedings, Second International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1987.

CHOW90 Chow F., Hennessy J.: „The Priority-Based Coloring Approach to Register Allocation". *ACM Transactions on Programming Languages*, October 1990.

CLAR85 Clark D., Emer J.: „Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement". *ACM Transactions on Computer Systems*, February 1985.

CLEM00 Clements A.: „The Udergraduate Curriculum in Computer Architecture". *IEEE Micro*, May/June 2000.

COHE81 Cohen D.: „On Holy Wars and a Plea for Peace". *Computer*, October 1981.

COLW85a Colwell R., Hitchock C., Jensen E., Brinkley-Sprunt H., Kollar C.: „Computers, Complexity and Controversy". *Computer*, September 1985.

COLW85b Colwell R., Hitchock C., Jensen E., Sprunt H.: „More Controversy About 'Computers, Complexity and Controversy'". *Computer*, December 1985.

COME95 Comeford R.: „An Overview of High Performance". *IEEE Spectrum*, April 1995.

COME00 Comeford R.: „Magnetic Storage: The Medium that Wouldn't Die". *IEEE Spectrum*, December 2000.

COOK82 Cook R., Dande N.: „An Experiment to Improve Operand Addressing". *Proceedings, Symposium on Architecture Support for Programming Languages and Operating Systems.* March 1982.

COON81 Coonen J.: „Underflow and Denormalized Numbers". *IEEE Computer*, March 1981.

COUT86 Coutant D., Hammond C., Kelley J.: „Compilers for the New Generation of Hewlett-Packard Computers". *Proceedings, COMPCON Spring '86*, March 1986.

CRAG79 Cragon H.: „An evaluation of Code Space Requirements and Performance of Various Architectures". *Computer Architecture News*, February 1979.

CRAG92 Cragon H.: *Branch Strategy Taxonomy and Performance Models.* Los Alamitos, IEEE Computer Society Press, 1992.

CRAW90 Crawford J.: „The i486 CPU: Executing Instructions in one Clock Cycle". *IEEE Micro*, February 1990.

CRIS97 Crisp R.: „Direct RAMBUS Technology: The New Main Memory Standard". *IEEE Micro*, November/December 1997.

DATT93 Dattatreya G.: „A Systematic Approach to Teaching Binary Arithmetic in a First Course". *IEEE Transactions on Education*, February 1993.

DAVI87 Davidson J., Vaughan R.: „The Effect of Instruction Set Complexity on Program Size and Memory Performance". *Proceedings, Second International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1987.

DENN68 Denning P.: „The Working Set Model for Program Behavior". *Communications of the ACM*, May 1968.

DEWA90 Dewar R., Smosna M.: *Microprocessors: A Programmer's View*. New York, McGraw-Hill, 1990.

DIJK63 Dijkstra E.: „Making an ALGOL Translator for the X1". *Annual Review of Automatic Programming*, t. 4, Pergamon, 1963.

DOET97 Doetting G. et al.: „S/390 Parallel Enterprise Server Generation 3: A Balanced System and Cache Structure". *IBM Journal of Research and Development*, July/September 1997.

DOWD98 Dowd K., Severance C.: *High Performance Computing*. Sebastopol, O'Reilly, 1998.

DUBE91 Dubey P., Flynn M.: „Branch Strategies: Modeling and Optimization". *IEEE Transactions on Computers*, October 1991.

DULO98 Dulong C.: „The IA-64 Architecture at Work". *Computer*, July 1998.

ECKE90 Eckert R.: „Communication Between Computers and Peripheral Devices – an Analogy". *ACM SIGSCE Bulletin*, September 1990.

ELAY85 El-Ayat K., Agarwal R.: „The Intel 80386 – Architecture and Implementation". *IEEE Micro*, December 1985.

EVEN00 Even G., Paul W.: „On the Design of IEEE Compliant Floating-Point Units". *IEEE Transactions on Computers*, May 2000.

EVER98 Evers M. et al.: „An Analysis of Correlation and Predictability: What Makes Two-Level Branch Prediction Work:. *Proceedings, 25th Annual International Symposium on Microarchitecture*, July 1998.

EVER01 Evers M., Yeh T.: „Understanding Branches and Designing Branch Predictors for High-Performance Microprocessors". *Proceedings of the IEEE*, November 2001.

FARM92 Farmwald M., Mooring D.: „A Fast Path to One Memory". *IEEE Spectrum*, October 1992.

FITZ81 Fitzpatrick D. at el.: „A RISCy Approach to VLSI". VLSI Design, 4. kw. 1981. Reprinted in *Computer Architecture News*, March 1982.

FLYN71 Flynn M., Rosin R.: „Microprogramming: An Introduction and a Viewpoint". *IEEE Transactions on Computers*, July 1971.

FLYN72 Flynn M.: „Some computer organizations and their effectiveness". *IEEE Transactions on Computers*, September 1972.

FLYN85 Flynn M., Johnson J., Wakefield S.: „On instruction sets and their formats". *IEEE Transactions on Computers*, March 1985.

FLYN87 Flynn M., Mitchell C., Mulder J.: „And Now a Case for More Complex Instruction Sets". *Computer*, September 1987.

FLYN01 Flynn M., Oberman S.: *Advanced Computer Arithmetic Design*. New York, Wiley, 2001.

FRAI83 Frailey D.: „Word Length of a Computer Architecture: Definitions and Applications". *Computer Architecture News*, June 1983.

FRIE96 Friedman M.: „RAID Keeps Going and Going and…". *IEEE Spectrum*, April 1996.

FURH87 Furht B., Milutinovic V.: „A Survey of Microprocessor Architectures for Memory Management". *Computer*, March 1987.

FUTR01 Futral W.: *InfiniBand Architecture: Development and Deployment*. Hillsboro, Intel Press, 2001.

GIFF87 Gifford D., Spector A.: „Case Study: IBM's System/360-370 Architecture". *Communications of the ACM*, April 1987.

GOLD91 Goldberg D.: „What Every Computer Scientist Should Know About Floating--Point Arithmetic". *ACM Computing Surveys*, March 1991.

HALF97 Halfhill T.: „Beyond Pentium II". *Byte*, December 1997.

HAND98 Handy J.: *The Cache Memory Book*. San Diego, Academic Press, 1993.

HAYE98 Hayes J.: *Computer Architecture and Organization*. New York, McGraw-Hill, 1998.

HEAT84 Heath J.: „Re-evaluation of RISC I". *Computer Architecture News*, March 1984.

HENN82 Hennessy J. et al.: „Hardware/Software Tradeoffs for Increased Performance". *Proceedings, Symposium on Architectural Support for Programming Languages and Operating Systems*, March 1982.

HENN84 Hennessy J.: „VLSI Processor Architecture". *IEEE Transactions on Computers*, December 1984.

HENN91 Hennessy J., Jouppi N.: „Computer Technology and Architecture: an Evolving Interaction". *Computer*, September 1991.

HENN96 Hennessy J., Patterson D.: *Computer Architecture; A Quantitative Approach*. San Mateo, Morgan Kaufmann, 1996.

HIDA90 Hidaka H., Matsuda Y., Asakura M., Kazuyasu F.: „The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory". *IEEE Micro*, April 1990.

HIGB90 Higbie L.: „Quick and Easy Cache Performance Analysis". *Computer Architecture News*, June 1990.

HILL64 Hill R.: „Stored Logic Programming and Applications". *Datamation*, February 1964.

HILL89 Hill M.: „Evaluating Associativity in CPU Caches". *IEEE Transactions on Computers*, December 1989.

HINT01 Hinton G. et al.: „The Microarchitecture of the Pentium 4 Processor". *Intel Technology Journal*, Q1 2001. http://developer.intel.com/technology/itj/

HUCK83 Huck T.: *Comparative Analysis of Computer Architectures*. Stanford University Technical Report No. 83-243, May 1983.

HUCK00 Huck J. et al.: „Introducing the IA-64 Architecture". *IEEE Micro*, September/October 2000.

HUGU91 Huguet M., Lang T.: „Architectural Support for Reduced Register Saving/Restoring in Single-Window Register Files". *ACM Transactions on Computer Systems*, February 1991.

HUTC96 Hutcheson G., Hutcheson J.: „Technology and Economics in the Semiconductor Industry". *Scientific American*, January 1996.

HWAN93 Hwang K.: *Advanced Computer Architecture*. New York, McGraw-Hill, 1993.

HWAN99 Hwang K. et al.: „Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space". *IEEE Concurrency*, January-March 1999.

HWU98 Hwu W.: „Introduction to Predicated Execution". *Computer*, January 1998.

HWU01 Hwu W., August D., Sias J.: „Program Decision Logic Optimization Using Predication and Control Speculation". *Proceedings of the IEEE*, November 2001.

IBM94 International Business Machines, Inc.: *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, Morgan Kaufmann, 1994.

IBM01 International Business Machines, Inc.: *64 Mb Synchronous DRAM*. IBM Data Sheet 364164, January 2001.

IEEE85 Institute of Electrical and Electronics Engineers. *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985, 1985.

INTE98 Intel Corp.: *Pentium Pro and Pentium II Processors and Related Products*. Aurora, 1998.

INTE00a Intel Corp.: *Intel IA-64 Architecture Software Developer's Manual (4 volumes)*. Dokument 245317 i 245320. Aurora, CO, 2000.

INTE00b Intel Corp.: *Itanium Processor Microarchitecture Reference for Software Optimization*. Aurora, Dokument 245473, CO, 2000.

INTE01a Intel Corp.: *IA-32 Intel Architecture Software Developer's Manual (2 volumes)*. Dokument 245470 i 245471, Aurora, CO, 2001.

INTE01b Intel Corp.: *Intel Pentium 4 Processor Optimization Reference Manual.* Dokument 248966-04, Aurora, CO, 2001.
http://developer.intel.com/design/pentium4/manuals/248966.htm.

JAME90 James D.: „Multiplexed Buses: The Endian Wars Continue". *IEEE Micro*, June 1990.

JARP01 Jarp S.: „Optimizing IA-64 Performance". *Dr. Dobb's Journal*, July 2001.

JOHN91 Johnson M.: *Superscalar microprocessor design*. Englewood Cliffs, Prentice-Hall, 1991.

JOUP88 Jouppi N.: „Superscalar Versus Superpipelined Machines". *Computer Architecture News*, June 1988.

JOUP89a Jouppi N., Wall D.: „Available instruction-level parallelism for superscalar and superpipelined machines". *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.

JOUP89b Jouppi N.: „The Nonuniform Distribution of Instruction-Level and Machine-Level Parallelism and Its Effect on Performance". *IEEE Transactions on Computers*, December 1989.

JTF01 Joint Task Force on Computing Curricula. *Computing Curricula 2001 Computer Science.* IEEE Computer Society and ACM, August 2001.

KAEL91 Kaeli D., Emma P.: „Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns". *Proceedings, 18th Annual International Symposium on Computer Architecture*, May 1991.

KAGA01 Kagan M.: „InfiniBand: Thinking Outside of the Box Design". *Communications System Design*, September 2001 (www.csdmag.com)

KANE92 Kane G., Heinrich J.: *MIPS RISC Architecture*. Englewood Cliffs, Prentice Hall, 1992.

KAPP00 Kapp C.: „Managing Cluster Computers". *Dr Dobb's Journal*, July 2000.

KATE83 Katevenis M.: *Reduced Instruction Set Computer Architectures for VLSI.* Rozprawa doktorska, Computer Science Department, University of California at Berkeley, October 1983, przedrukowane przez MIT Press, Cambridge, 1985.

KATH01 Kathail B., Schlansker M., Rau B.: „Compiling for EPIC Architectures". *Proceedings of the IEEE*, November 2001.

KATZ89 Katz R., Gibson G., Patterson D.: „Disk System Architecture for High Performance Computing". *Proceedings of the IEEE*, December 1989.

KEET01 Keeth B., Baker R.: *DRAM Circuit Design: A Tutorial.* Piscataway, IEEE Press, 2001.

KHUR01 Khurshudov A.: *The Essential Guide to Computer Data Storage.* Upper Saddle River, Prentice Hall, 2001.

KNUT71 Knuth D.: „An Empirical Study of FORTRAN Programs". *Software Practice and Experience*, t. 1, 1971.

KNUT98 Knuth D.: *Sztuka programowania. Tom. 2: Algorytmy seminumeryczne*. WNT, Warszawa 2002.

KUCK72 Kuck D., Muraoka Y., Chen S.: „On the Number of Operations Simultaneously Executable in Fortran-Like Programs and Their Resulting Speedup". *IEEE Transactions on Computers*, December 1972.

KUGA91 Kuga M., Murakami K., Tomita S.: „DSNS (Dynamically-hazard Resolved, Statically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture". *Computer Architecture News*, June 1991.

LEE91 Lee R., Kwok A., Briggs F.: „The Floating-Point Performance of a Superscalar SPARC Processor". *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

LILJ88 Lilja D.: „Reducing the Branch Penalty in the Pipelined Processors". *Computer*, July 1988.

LILJ93 Lilja D.: „Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons". *ACM Computing Surveys*, September 1993.

LOVE96 Lovett T., Clapp R.: „Implementation and Performance of a CC-NUMA System". *Proceedings, 23rd Annual International Symposium on Computer Architecture*, May 1996.

LUND77 Lunde A.: „Empirical Evaluation of Some Features of Instruction Set Processor Architectures". *Communications of the ACM*, March 1977.

LYNC93 Lynch M.: *Microprogrammed State Machine Design*. Boca Raton, 1993.

MACG84 MacGregor D., Mothersole D., Moyer B.: „The Motorola MC68020". *IEEE Micro*, August 1984.

MAHL94 Mahlke S. et al.: „Characterizing the Impact of Predicated Execution on Branch Prediction". *Proceedings, 27th International Symposium on Microarchitecture*, December 1994.

MAHL95 Mahlke S. et al.: „A Comparison of Full and Partial Predicated Execution Support for ILP Processors". *Proceedings, 22nd International Symposium on Computer Architecture*, June 1995.

MAK97 Mak P. et al.: „Shared-Cache Clusters in a System with Fully Shared Memory". *IBM Journal of Research and Development*, July/September 1997.

MALL75 Mallach E.: „Emulation Architecture". *Computer*, August 1975.

MALL83 Mallach E., Sondak N.: *Advances in Microprogramming.* Dedham, Artech House, 1983.

MANJ01a Manjikian N.: „More Enhancements of the SimpleScalar Tool Set". *Computer Architecture News*, September 2001.

MANJ01b Manjikian N.: „Multiprocessor Enhamcements of the SimpleScalar Tool Set". *Computer Architecture News*, March 2001.

MANO01 Mano M.: *Logic and Computer Design Fundamentals.* Upper Saddle River, Prentice Hall, 2001.

MARC90 Marchant A.: *Optical Recording.* Reading, Addison-Wesley, 1990.

MARK00 Markstein P.: *IA-64 and Elementary Functions.* Upper Saddle River, Prentice Hall PTR, 2000.

MASH95 Mashey J.: „CISC vs. RISC (or what is RISC really)". *USENET comp.arch newsgroup, article 46782*, February 1995.

MASS97 Massiglia P.: *The RAID Book: A Storage System Technology Handbook.* St. Peter, The Raid Advisory Board, 1997.

MAYB84 Mayberry W., Efland G.: „Cache Boosts Multiprocessor Performance". *Computer Design*, November 1984.

MCEL85 McEliece R.: „The Reliability of Computer Memories". *Scientific American*, January 1985.

MEE96a Mee C., Daniel E. (ed.): *Magnetic Recording Technology.* New York, McGraw-Hill, 1996.

MEE96b Mee C., Daniel E. (ed.): *Magnetic Storage Handbook.* New York, McGraw-Hill, 1996.

MILE00 Milenkovic A.: „Achieving High Performance in Bus-Based Shared-Memory Multiprocessors", *IEEE Concurrency*, July-September 2000.

MIRA92 Mirapuri S., Woodacre M., Vasseghi N.: „The MIPS R4000 Processor". *IEEE Micro*, April 1992.

MOOR65 Moore G.: „Cramming More Components Onto Integrated Circuits". *Electronics Magazine*, April 19, 1965.

MORS78 Morse S., Pohlman W., Ravenel B.: „The Intel 8086 Microprocessor: A 16-bit Evolution of the 8080". *Computer*, June 1978.

MOSH01 Moshovos A., Sohi G.: „Microarchitectural Innovations: Boosting Microprocessor Performance Beyond Semiconductor Technology Scaling". *Proceedings of the IEEE*, November 2001.

MOTO01 Motorola, Inc.: *PowerPC MPC7410 RISC Microprocessor Hardware Specifications.* Denver, 2001. www.motorola.com

MYER78 Myers G.: „The Evaluation of Expressions in a Storage-to-Storage Architecture". *Computer Architecture News*, June 1978.

NAYF96 Nayfeh B., Olukotun K., Singh J.: „The Impact of Shared Cache Clustering in Small-Scale Shared-Memory Multiprocessors". *Proceedings of the Second International Symposium on High Performance Computer Architecture*, 1996.

NOVI93 Novitsky J., Azimi M., Ghaznavi R.: „Optimizing Systems Performance Based on Pentium Processors". *Proceedings COMPCON '92*, February 1993.

OBER97a Oberman S., Flynn M.: „Design Issues in Division and Other Floating-Point Operations". *IEEE Transactions on Computers*, February 1997.

OBER97b Oberman S., Flynn M.: „Division Algorithms and Implementations". *IEEE Transactions on Computers*, August 1997.

OVER01 Overton M.: *Numerical Computing with IEEE Floating Point Arithmetic.* Philadelphia, Society for Industrial and Applied Mathematics, 2001.

PADE81 Padegs A.: „System/360 and Beyond". *IBM Journal of Research and Development*, September 1981.

PADE88 Padegs A., Moore B., Smith R., Buchholz W.: „The IBM System/370 Vector Architecture: Design Considerations". *IEEE Transactions on Communications*, May 1988.

PARH00 Parhami B.: *Computer Arithmetic: Algorithms and Hardware Design.* Oxford, Oxford University Press, 2000.

PARK89 Parker A., Hamblen J.: *An introduction to Microprogramming with Exercises Designed for the Texas Instruments SN74ACT8800 Software Development Board*. Dallas, Texas Instruments, 1989.

PATT82a Patterson D., Sequin C.: „A VLSI RISC". *Computer*, September 1982.

PATT82b Patterson D., Piepho R.: „Assessing RISCs in High-Level Language Support". *IEEE Micro*, November 1982.

PATT84 Patterson D.: „RISC Watch". *Computer Architecture News*, March 1984.

PATT85a Patterson D.: „Reduced Instruction Set Computers". *Communications of the ACM*, January 1985.

PATT85b Patterson D., Hennessy J.: „Response to 'Computers, Complexity and Controversy'". *Computer*, November 1985.

PATT88 Patterson D., Gibson G., Katz R.: „A Case for Redundant Arrays of Inexpensive Disks (RAID)". *Proceedings, ACM SIGMOND Conference of Management of Data*, June 1988.

PATT98 Patterson D., Hennessy J.: *Computer Organization and Design: the Hardware/Software Interface*. San Mateo, Morgan Kaufmann. 1998.

PATT01 Patt Y.: „Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution". *Proceedings of the IEEE*, November 2001.

PEIR99 Peir J., Hsu W., Smith A.: „Functional Implementation Techniques for CPU Cache Memories". *IEEE Transactions on Computers*, February 1999.

PELE97 Peleg A., Wilkie S., Weiser U.: „Intel MMX for Multimedia PCs". *Communications of the ACM*, January 1997.

PFIS98 Pfister G.: *In Search of Clusters.* Upper Saddle River, Prentice Hall, 1998.

POPE91 Popescu V. et al.: „The Metaflow Architecture". *IEEE Micro*, June 1991.

POTT94 Potter T. et al.: „Resolution of Data and Control-Flow Dependencies in the PowerPC 601". *IEEE Micro*, October 1994.

PRES01 Pressel D.: „Fundamental Limitations on the Use of Prefetching and Stream Buffers for Scientific Applications". *Proceedings, ACM Symposium on Applied Computing*, March 2001.

PRIN91 Prince B.: *Semiconductor Memories*. New York. Wiley, 1991.

PRIN99 Prince B.: *High Performance Memories: New Architecture DRAMs and SRAMs, Evolution and Function.* New York, Wiley, 1999.

PRZY88 Przybylski S., Horowitz M., Hennessy J.: „Performance trade-offs in Cache Design". *Proceedings, Fifteenth Annual International Symposium on Computer Architecture*, June 1988.

PRZY90 Przybylski S.: „The Performance Impact of Block Size and Fetch Strategies". *Proceedings, 17th Annual International Symposium on Computer Architecture*, May 1990.

RADI83 Radin G.: „The 801 Minicomputer". *IBM Journal of Research and Development*, May 1983.

RAGA83 Ragan-Kelley R., Clark R.: „Applying RISC Theory to a Large Computer". *Computer Design*, November 1983.

RAUS80 Rauscher T., Adams P.: „Microprogramming: A Tutorial and Survey of Recent Developments". *IEEE Transactions on Computers*, January 1980.

RECH98 Reches S., Weiss S.: „Implementation and Analysis of Path History in Dynamic Branch Prediction Schemes". *IEEE Transactions on Computers*, August 1998.

RODR01 Rodriguez M., Perez J., Pulido J.: „An Educational Tool for Testing Caches on Symmetric Multiprocessors". *Microprocessors and Microsystems*, June 2001.

ROSC99 Rosch W.: *The Winn L. Rosch Hardware Bible*. Indianapolis, Sams, 1999.

SATY81 Satyanarayanan M., Bhandarkar D.: „Design Trade-offs in VAX-11 Translation Buffer Organization". *Computer*, December 1981.

SCHA97 Schaller R.: „Moore's Law: Past, Present, and Future". *IEEE Spectrum*, June 1997.

SCHL00a Schlansker M., Rau B.: „EPIC: Explicitly Parallel Instruction Computing". *Computer*, February 2000.

SCHL00b Schlansker M., Rau B.: *EPIC: An Architecture for Instruction-Level parallel Processors.* HPL Technical Report HPL-1999-111, Hewlett-Packard Laboratories (www.hpl.hp.com), February 2000.

SCHW99 Schwarz E., Krygowski C.: „The S/390 G5 Floating-Point Unit". *IBM Journal of Research and Development*, September-November 1999.

SEBE76 Sebern M.: „A Minicomputer-compatible Microcomputer System: The DEC LSI-11". *Proceedings of the IEEE*, June 1976.

SEGE91 Segee B., Field J.: *„Microprogramming and Computer Architecture*. New York, Wiley, 1991.

SERL86 Serlin O.: „MIPS, Dhrystones, and Other Tales". *Datamation*, June 1, 1986.

SHAN38 Shannon C.: „Symbolic Analysis of Relay and Switching Circuits". *AIEE Transactions*, Vol. 57, 1938.

SHAN95a Shanley T., Anderson D.: *PCI Systems Architecture*. Richardson, Mindshare Press, 1995.

SHAN95b Shanley T.: *PowerPC System Architecture*. Reading, Addison-Wesley, 1995.

SHAN98 Shanley T.: *Pentium Pro and Pentium II System Architecture*. Reading, Addison--Wesley, 1998.

SHAR97 Sharma A.: *Semiconductor Memories: Technology, Testing, and Reliability*. New York, IEEE Press, 1997.

SHAR00 Sharangpani H., Arona K.: „Itanium Processor Microarchitecture". *IEEE Micro*, September/October 2000.

SHER84 Sherburne R.: *Processor Design Tradeoffs in VLSI*. Tezy doktorskie, Report No. UCB/CSD 84/173, University of California at Berkeley, April 1984.

SIEW82 Siewiorek B., Bell C., Newell A.: *Computer Structures: Principles and Examples*. New York, McGraw-Hill, 1982.

SIMA97 Sima D.: „Superscalar Instruction Issue". *IEEE Micro*, September/October 1997.

SIMO69 Simon H.: *The Sciences of Artificial*. Cambridge, MIT Press, 1969.

SMIT82 Smith A.: „Cache memories". *ACM Computing Surveys*, September 1982.

SMIT87 Smith A.: „Line (Block) Size Choice for CPU Cache Memories". *IEEE Transactions on Communications*, September 1987.

SMIT89 Smith M., Johnson M., Horowitz M.: „Limits on Multiple Instruction Issue". *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.

SMIT95 Smith J., Sohi G.: „The Microarchitecture of Superscalar Processors". *Proceedings of the IEEE*, December 1995.

SODE96 Soderquist P., Leeser M.: „Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementations". *ACM Computing Surveys*, September 1996.

SOHI90 Sohi G.: „Instruction Issue Logic for High-Performance Interruptable, Multiple Functional Unit, Pipelined Computers". *IEEE Transactions on Computers*, March 1990.

STAL00 Stallings W.: *Data and Computer Communications*. Wyd. 5. Upper Saddle River, Prentice Hall, 1994.

STAL01 Stallings W.: *Operating Systems, Internals and Design Principles*. Wyd. 4. Upper Saddle River, Prentice Hall, 1995.

STEN90 Stenstrom P.: „A Survey of Cache Coherence Schemes of Multiprocessors". *Computer*, June 1990.

STEV64 Stevens W.: „The Structure of System/360, Part II: System Implementation". *IBM Systems Journal*, Vol. 3, Nr 2, 1964. Przedruk w [BELL71A] i [SIEW82].

STON93 Stone H.: *High-Performance Computer Architecture*. Reading, Addison-Wesley, 1993.

STRE78 Strecker W.: „VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family". *Proceedings, National Computer Conference*, 1978.

STRE83 Strecker W.: „Transient Behavior of Cache Memories". *ACM Transactions on Computer Systems*, November 1983.

STRI79 Stritter E., Gunter T.: „A Microprocessor Architecture for a Changing World: The Motorola 68000". *Computer*, February 1979.

SWAR90 Swartzlander E. (ed.): *Computer Arithmetic, Vol. I i II. Los Alamitos*, IEEE Computer Society Press, 1990.

TABA91 Tabak D.: *Advanced Microprocessors*. New York, McGraw-Hill, 1991.

TAMI83 Tamir Y., Sequin C.: „Strategies for Managing the Register File in RISC". *IEEE Transactions on Computers*, November 1983.

TANE78 Tanenbaum A.: „Implications of Structured Programming for Machine Architecture". *Communications of the ACM*, March 1978.

TANE99 Tanenbaum A.: *Structured Computer Organization*. Englewood Cliffs, Prentice-Hall, 1999.

THOM94 Thompson T., Ryan B.: „PowerPC 620 Soars". *Byte*, November 1994.

THOM00 Thompson D.: „IEEE 1394: Changing the Way We Do Multimedia Communications". *IEEE Multimedia*, April-June 2000.

TI90 Texas Instruments Inc.: *SN74ACT880 Family Data Manual*. SCSS006C, 1990.

TJAD70 Tjaden G., Flynn M.: „Detection and Parallel Execution of Independent Instructions". *IEEE Transactions on Computers*, October 1970.

TOMA93 Tomasevic M., Milutinovic V.: *The Cache Coherence Problem Shared-Memory Multiprocessors: Hardware Solutions*. Los Alamitos, IEEE Computer Society Press, 1993.

TOON81 Toong H., Gupta A.: „An Architectural Comparison of Contemporary 16-bit Microprocessors". *IEEE Micro*, May 1981.

TRIE01 Triebel W.: *Itanium Architecture for Software Developers*. Intel Press, 2001.

TUCK67 Tucker S.: „Microprogram Control for System/360". *IBM Systems Journal*, nr 4, 1967. Również w [MALL83].

TUCK87 Tucker S.: „The IBM 3090 System Design with Emphasis on the Vector Facility". *Proceedings, COMPCON Spring '87*, February 1987.

VOEL88 Voelker J.: „The PDP-8". *IEEE Spectrum*, November 1988.

VOGL94 Vogley B.: „800 Megabyte Per Second Systems Via Use of Synchronous DRAM". *Proceedings, COMPCON '94*, March 1994.

VONN45 Von Neumann J.: *First Draft of a Report on the EDVAC*. Moore School, University of Pennsylvania, 1945. Reprinted in *IEEE Annals on the History of Computing*, No. 4, 1993.

VRAN80 Vranesic Z., Thurber K.: „Teaching Computer Structures". *Computer*, June 1980.

WALL85 Wallich P.: „Toward Simpler, Faster Computers". *IEEE Spectrum*, August 1985.

WALL91 Wall D.: „Limits of Instruction-Level Parallelism". *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

WANG99 Wang G., Tafti D.: „Performance Enhancement on Microprocessors with Hierarchical Memory Systems for Solving Large Sparse Linear Systems". *International Journal of Supercomputing Applications*, vol. 13, 1999.

WARD90 Ward S., Halstead R.: *Computation Structures*. Cambridge, MIT Press, 1990.

WEIN75 Weinberg G.: *An Introduction to General Systems Thinking*. New York, Wiley, 1975.

WEIS84 Weiss S., Smith J.: „Instruction Issue Logic in Pipelined Supercomputers". *IEEE Transactions on Computers*, November 1984.

WEIS94 Weiss S., Smith J.: *POWER and PowerPC*. San Francisco, Morgan Kaufmann, 1994.

WEYG01 Weygant P.: *Clusters for High Availability*. Upper Saddle River, Prentice Hall, 2001.

WHIT97 Whitney S. et al.: „The SGI Origin Software Environment and Application Performance". *Proceedings, COMPCON Spring '97*, February 1997.

WICK97 Wickelgren I.: „The Facts About FireWire". *IEEE Spectrum*, April 1997.

WILK51 Wilkes M.: „The Best Way to Design an Automatic Calculating Machine". *Proceedings, Manchester University Computer Inaugural Conference*, July 1951.

WILK53 Wilkes M., Stringer J.: „Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer". *Proceedings of the Cambridge Philosophical Society*, April 1953. Reprinted in [SIEW82].

WILL90 Williams F., Steven G.: „Address and Data Register Separation on the M68000 Family". *Computer Architecture News*, June 1990.

YEH91 Yeh T., Patt Y.: „Two-Level Adaptative Training Branch Prediction". *Proceedings, 24th Annual International Symposium on Microarchitecture*, 1991.

ZHAN01 Zhang Z., Zhu Z., Zhang X.: „Cached DRAM for ILP Processor Memory Access Latency Reduction". *IEEE Micro*, July-August 2001.

Skorowidz

A

ACM witryna WWW 36
adres, dekodowanie 787
- efektywny 311–313, 322
- fizyczny 297–298
–, generowanie 676–678
- logiczny 297–298, 306
–, rejestr podstawowy 433
- wirtualny 306
adresowania tryb 427–458
adresowanie bezpośrednie 430–431
- bezwzględne 441–443
- natychmiastowe 430
- pośrednie 431, 441–443
- - indeksowane 438
–, PowerPC 441–443
–, SMP 716
- stosu 434
- z przesunięciem 432–433
adresu linie 95
- pole 675–676
- rejestr 431–432, 465
- zakres 445
akumulator 44, 79
ALAT w IA-64 619
algebra Boole'a 766–768, 791
algorytm Bootha 339–342
- Dijkstry 419
AND, funkcja logiczna 387
antyzależność 570–571
arbitraż PCI 113–115
- SMP 716
architektura CISC 515–516
- Hewlett-Packard PA-RISC 600
architektura IBM S/370 25, 752–753
- IBM S/390 349, 753
- klastrowa 736–737
- RISC 542
- von Neumanna 75
- wektorowa 756–758
argument, liczba 444
–, obliczanie adresu 81
–, pobieranie 82
–, rodzaje 377–379
arytmetyczne przesunięcie 339
arytmetyczno-logiczna jednostka 31, 39, 322–323
arytmetyka całkowitoliczbowa 330–345
- komputera 322–363
- - IAS 45
- zmiennopozycyjna 322, 351–362
ASCII rozkaz maszynowy 379
asembler 410
asemblerowy język 408–410
asynchroniczne operacje magistralowe 103
automatyczne indeksowanie 434

B

bajtów porządkowanie 421–425
bardzo duży stopień scalenia (VLSI) 528–530
Berkeley RISC komputery 520
bezpośredni dostęp do pamięci (DMA) 91, 93, 124, 226, 235, 248–253
bezpośrednie odwzorowanie 133–134, 137–138
bezwarunkowe rozgałęzienie 45
bezwzględna skalowalność klastrów 732
- wartość 386
bieżąca ramka, znacznik (CFM) 627–629
bieżące okno, wskaźnik (CWP) 520, 547

binarne dodawanie 791
– dzielenie 343
– mnożenie 337–338
binarny system liczbowy 323, 808–809
bitów porządkowanie 425
błąd strony 298
błyskawiczna pamięć 172
Boole'a algebra 766–768, 791
boolowskich wyrażeń upraszczanie 773–782
boolowskie funkcje, forma kanoniczna 776
– –, implementacja 771–782
– operatory 767
Bootha algorytm 339–342
bramki logiczne 769
– – NAND 770
– – NOR 770
BTB 581

C

całkowite liczby, arytmetyka 330–345
– –, reprezentacja 323
CAV 194–195, 215
CC-NUMA 740–743
CD-R 217
CD-ROM 214–216
CD-RW 214, 217
CFM 627–629
ciche i sygnalizujące NaN 359–360
CICS komputer 525–527
cienkokońcowe struktury 370, 421–425
CLV 215
Cray superkomputery 750
CWP 520, 547
cykl maszynowy 527, 655
–, podkradanie 252
– pośredni 470
cylindry 199
czas dostępu do dysku 200
– – do pamięci 124–129
czasowa lokalność 158–159

D

D przerzutnik 795
dane, formatowanie 193–195
–, kanały 49
–, linie 95
–, organizacja 193–195
–, magistrala 95
–, przepływ 472–473
–, przetwarzanie 26–27
–, rejestry 465
dane, spekulacja 600, 619–620
–, strumień 711–713
–, ścieżki 649
–, zapisywanie 26–27
DDR-SDRAM 185
DED kod 181–183
dekodery 669
dekrementacja 388
DeMorgana prawo 769, 782
demultiplekser 787
Dijkstry algorytm 420
długookresowa kolejka 290
dodatni niedomiar 347
dodatnie przepełnienie 347
dodawanie, schemat blokowy 336
– w arytmetyce zmiennopozycyjnej 353, 355
– w uzupełnieniu do dwóch 331–334
DRAM 62, 122, 166, 169–170, 183–188
druga generacja komputerów 48
DVD 217–218
DVD-R 214
DVD-ROM 219
DVD-RW 214
dwuetapowy potok rozkazów 476
dwukońcowe struktury 370, 421–425
dwupoziomowa pamięć 156–157
dynamiczna strategia rozgałęziania 485
dysk, czas wyszukiwania 200–201
– dwustronny 198
– elastyczny 199
–, formatowanie 195–196
– jednostronny 198
– kompaktowy 214–215
– magnetyczny 192–204
–, mechanizmy głowic 198
–, napęd 232
–, opóźnienie rotacyjne 200–203
– optyczny 122, 214
–, organizacja sekwencyjna 203
–, pamięć podręczna 129, 157
–, parametry 200–204
–, układ danych 194
– Winchester 197, 199
dyskietka 199
dyspozytorska jednostka 586–589
dzielenie liczb całkowitych 342–351
– zmiennopozycyjne 355–357
dziesiętny system 808

E

EBCDIC 379
EDVAC 39

EEPROM 168, 172
EFLAGS rejestr 492
ENIAC 38
EPROM 168, 172, 175

F

FIFO, pamięć podręczna 143
FireWire 226, 257–262, 301
fizyczna warstwa FireWire 258–260
fizyczne rekordy 220
fizyczny adres 297–298
FORK 746
funkcje boolowskie 77–782
– logiczne 387

G

generacje komputerów 48
grafy, kolorowanie 524–525

H

Hamminga kod 178–179, 182
historia komputerów 38–59

I

IA-64 599–630
IAS komputer 41–45
IBM 360/91 483
IBM 700/7000 50
IBM 3033 679, 690–691
IBM 3090 751–758
IBM 7094 49,51
IBM S/360 53–55, 664, 679
IBM S/370 385–386, 752–753
IBM S/390 349, 362, 723, 753
IEEE standard zmiennopozycyjny 350
iloczyn sum (POS) 772–773
indeksowanie 433–434
indeksu rejestry 434, 465
InfiniBand 226, 263–266
inkrementacja 386
Intel 80486 489–499
– 82C55A 245–247
–, ewolucja mikroprocesorów 58–59
interfejs, rodzaje 256–257
IRA 230–231
Itanium 601–602, 629–631
izolowane wejście-wyjście 238

J

JCL 279
jednopoziomowa pamięć 156–157
jednorodny dostęp do pamięci (UMA) 740
jednostka sterująca 31, 34
języki wysokiego poziomu 514, 526
J-K przerzutnik 797–798
JOIN 746

K

kanałowa architektura wejścia-wyjścia 255
Karnaugha mapy 773–776
katalogowe protokoły 727
k-drożna sekcyjno-skojarzeniowa organizacja 140–142
klaster 732–739
kolejka pośrednia 272
kombinacyjne układy 771–772
kompaktowe dyski 214–215
komputer, organizacja 24, 713, 751
komputery, rodzina 54–55
konflikt zasobów 566
konwersja liczb binarnych na dziesiętne 809–812
korekcja błędu, kod 178
krótkookresowa kolejka 290

L

lampy próżniowe 38–48
LFU pamięć podręczna 143
liczby zdenormalizowane 359
licznik rozkazów (PC) 43, 77, 433, 640–642
liczniki synchroniczne 801–803
–, układy sekwencyjne 800–803
LIFO 434
logiczne adresy 297–298, 308
– bramki 768
– funkcje 387
– przesunięcie 387
lokalność odniesienia 128
LRU pamięć podręczna 141
LSI-11 687–689

Ł

łącza warstwa 260

M

macierze, mnożenie 745–746
magistrala, arbitraż 244

magistrala, architektura 96–99
–, cykl 101
–, działanie 96
–, hierarchia 97–99
–, koordynacja asynchroniczna 102
–, – synchroniczna 101
–, linie sterowania 95–96
– magistrala procesora wewnętrzna 651
–, projektowanie 100
–, przebiegi czasowe 118
– systemowa 94, 103, 648
–, szerokość 103–105
–, typy transferu danych 104
magnetooporowy czujnik 193
magnetooptyczna pamięć 192
magnetyczna taśma 220–221
magnetyczne dyski 192–204
mały stopień scalenia (SSI) 52–53, 788
mantysa 346
maszynowe rozkazy 370–377
maszynowy cykl 527, 655
– paralelizm 566–567, 572–573
MC6800 469
MESI protokół 728–732
mieszanie, funkcja 299–300
międzyrekordowe przerwy 220
mikrodiagnostyka 704
mikroelektronika 51–52
mikrooperacje 639–645
mikroprocesor 57–62, 468–469, 532
mikroprogram 661, 665, 671–673, 704–705
mikroprogramowane sterowanie 664–704
mikrorozkaz, format 665–666
–, kodowanie 684–687
–, szeregowanie 673–680
–, taksonomia 681–684
–, wykonywanie 673, 680–691
mikrosekwenser 695–700
MIMD 711–713
MIPS 537–546
MISD 711–713
mnożenie zmiennopozycyjne 355–357
–, uzupełnienie do dwóch 334–342
modułowa organizacja pamięci głównej 176–178
monitor 278
Moore'a prawo 53
multiplekser 782–784

N

NAND 770, 782
NaT bit 613
natychmiastowe adresowanie 430
negacja 325, 330, 386
niejednorodny dostęp do pamięci (NUMA) 740–743
normalizacja zmiennopozycyjna 355
NOOP 535–536
NOR bramki 770, 782
notacja wrostkowa 419
NUMA 740–743

O

obliczanie wektorowe 744–758
obraz, element 403
ochronne bity 357
odczyt-modyfikacja-zapis 105
odejmowanie, schemat blokowy 336
–, uzupełnienie do dwóch 331–333
odwzorowane w pamięci wejście-wyjście 237–238
odwzorowanie *k*-drożne sekcyjno-skojarzeniowe 140–141
– skojarzeniowe 138, 140
okna rejestrów 519–520
opóźnienie rotacyjne 200–203
–, szczelina 535
opóźnione rozgałęzienie 487–488, 535–536
optyczny dysk 122
optymalizowane rozgałęzienie opóźnione 535
OR 387
organizacja wieloprocesorowa 512, 711–713
ortogonalność 447–448
ośmiu hetmanów problem 615

P

pamięć, czas cyklu 49, 124–125
– dwupoziomowa 156–157
– główna 31, 76
–, hierarchia 122, 125–129
– jednopoziomowa 156–157
–, komórka 51, 167
– magnetooptyczna 192
–, moduł 76
–, przegląd systemów 122–129
–, pojemność 125–129
– podręczna 121–152
– –, algorytm zastępowania 141
– –, dyskowa 129, 157
– – *k*-drożna sekcyjno-skojarzeniowa 142
– –, polityka zapisu 144–145
– –, odwzorowanie 133, 135
– –, Pentium 4, 581–582

pamięć podręczna, PowerPC 152
– –, projektowanie 132–147
– –, protokół MESI 723–732
– –, rozmiar 133
– –, – wiersza 145
– –, spójność 723–728
– – wewnętrzna 150
– – wielopoziomowa 145–146,
– – w pełni skojarzeniowa 139
– – zewnętrzna 146
– rdzeniowa 57
–, rejestr adresu (MAR) 43, 76–77, 466, 640–641
–, – buforowy (MBR) 43, 76–77, 466, 640–641
– skojarzeniowa 124
–, system 122–125
–, technologia optyczna 192
– wewnętrzna 123
– wieloportowa 718
– wirtualna 128, 157, 298–305
–, zarządzanie 283
pamięci podręczne wielopoziomowe 145–146
– – zewnętrzne 146
partycjonowanie 292–297
PCI, arbiter magistrali 113
–, linie sygnałowe 107–109
–, rozkazy 107–111
PDP-10 447–448
PDP-11 80–81, 448–450, 515, 679
PDP-8 55–56, 445–447
Pentium 4 576–584
Pentium II 306–310, 493, 495
Pentium, adresowanie względne 438
–, ewolucja 65–67
–, format rozkazu 453–455
–, instrukcje wywołania-powrotu 398
–, kod operacji 453
–, kody warunkowe 400–402
Pentium MMX 402, 404, 496–497
–, organizacja rejestrów 490–491
–, prefiksy rozkazu 453
Pentium Pro 601–602
–, procesor przerwań 498
–, przerwania 497
–, rejestry sterowania 494
–, rodzaje danych 380–382
–, rozmiar argumentu 453
–, tryb natychmiastowy 435
–, tryby adresowania 435–438
–, tryb z przemieszczeniem 437
–, zarządzanie pamięcią 400
peryferyjne urządzenie 227
pętli bufor 483–484
pierścieniowy stos 548
piksel 403
pionowe mikrorozkazy 665–666, 669, 681
PLA 789–790
pobieranie, cykl 44, 472, 640–642
–, jednostka 147
–, mechanizm 629
–, rozkaz 462, 476
podglądania protokoły 727–728
podstawa normalizacyjna 355
polska notacja odwrotna 419
POP 416–417
pośredni cykl 470
pośrednia kolejka 292
pośrednie adresowanie 431
– indeksowane adresowanie 438
potok, działanie 478, 578–579, 749–750
–, etap 545
–, opis 475
–, optymalizacja 535–537
– rozkazów, dwuetapowy 476
– skalarny 565
–, wydajność 480
PowerPC 601 584–589
PowerPC 620 590–592
–, ewolucja 65–67
–, formaty rejestrów 503
–, – rozkazu 454–456
PowerPC G4 151
–, organizacja pamięci podręcznej 150–152
–, procesor 500–507
–, rejestry widzialne dla użytkownika 501
–, rodzaje danych 382
–, – operacji 408
–, tablica przerwań 505
–, tryby adresowania 438–441
–, wewnętrzne pamięci podręczne 150
–, zarządzanie pamięcią 309–313
poziome mikrorozkazy 665–666, 681, 684
półprzewodnikowa pamięć 57, 128, 166–177
półsłowo 382
prawdy tablica 767, 771
prawdziwa zależność danych 564–566
predykacja 604–624
problem ośmiu hetmanów 615
proceduralne zależności 566
procedury, wywołanie 394–397, 516
–, zagnieżdżenie 395
procesor, ewolucja 63
–, kategorie 370
–, rejestr stanu (PSR) 547
–, struktura i działanie 462–507
–, superskalarny 560

procesor, własności 533
–, wymagania funkcjonalne 638
procesory wektorowe 750
programowalna tablica logiczna (PLA) 787–790
programowane wejście-wyjście 226, 235–240, 248–253
PROM 168–172
przejmowanie danych 736
przeplatane wieloprogramowanie 714
przerwa powietrzna 199
przerwanie 82–90
–, cykl 83
–, przetwarzanie 241–243, 497–499
– wektorowe 82–90
przerwaniami sterowane wejście-wyjście 226, 235, 240–248
przerzutniki D 795
– J-K 796, 797
–, układy sekwencyjne 793–794, 798
przestrzenna lokalność 158
przesunięta reprezentacja zmiennopozycyjna 346
przetwarzanie równoległe 709–758
– wsadowe 278–281
– zagnieżdżonych przerwań 90
przyrostkowa notacja 419
przywracanie 736
pseudorozkaz 409
PSW 242, 462, 467
PUSH 416–417
Pyramid, komputer 520

Q

Quine'a-McKluskeya tablice 778–782

R

R3000 544–545
R4000 537–546
RAID, konfiguracja 205
–, poziomy 205–206, 209–213
Rambus DRAM 183, 185–190
ramka 111, 297, 398
ramki stron 297
RDRAM 183, 185–190
rdzeniowa pamięć 57
rejestry, adresowanie 431–432
–, organizacja 464–469
–, predykacji 624
–, procesora 372, 418, 463
–, przemianowanie 571–572
rejestry robocze 465
– widzialne dla użytkownika 462
– zamienne 583
rekordy fizyczne 220
reprezentacja zmiennopozycyjna 322, 345–351
resztkowe sterowanie 679
rezydentny monitor 278
RISC, komputery superskalarne 605
–, procesory 560
–, przetwarzanie potokowe 534–537
–, uzasadnienie 601
rodzina komputerów 54–55
–, koncepcja 512
ROM 168–173, 789, 791–792
rotacyjne opóźnienie 200–203
rozgałęzienie, bufor celów 581
–, diagram stanów 487
– opóźnione 487–488, 535–536
–, przewidywanie 484–487
–, strategia 485
rozkaz, cykl 44, 78, 470–474, 655
–, długość 443
–, format 372, 443–453
–, okno 570
–, polityka wydawania 567–569
–, potok 462, 475–490
–, rejestr 43, 77–78
–, słowo 42
–, strumień 712
–, wstępne pobieranie 475
–, zbiór 369–425
rozkazy maszynowe 370–377
– o zmiennej długości 448–453
–, rejestr buforowy (IBR) 43
rozszczepiona pamięć podręczna 146–147
równoległe przetwarzanie 709–758

S

SCSI magistrala systemowa 97
SDRAM 183–186
SEC kod 181–182
segmenty, deskryptory 308–310
–, liczba 307
–, wskaźniki 465
sekcyjno-skojarzeniowe odwzorowanie 138–140
sektory 194
sekwencyjny dostęp do pamięci 142
semantyczna luka 514
serpentynowa rejestracja 220
sieć lokalna i rozległa 97
SimpleScalar 817

skalarne przetwarzanie 745–746
skalarny potok 565
skalowalna architektura procesora (SPARC) 520, 546–552
skojarzeniowa pamięć 124
skojarzeniowe odwzorowanie 138
skok, rozkaz 535
słowo 42, 123, 444
– stanu programu (PSW) 242, 462, 467
SMPCache 817
SOP 771–772
SPARC 520, 546–552
spekulacja i predykacja 616
S-R przerzutnik 794–796
SSI 52–53, 788
standard zmiennopozycyjny IEEE 359
sterowanie, linie 95
–, mikrooperacje 649
–, pamięć 666–667
– procesorem 645–658
– – Pentium 491
–, rejestr adresu 667
–, – buforowy 667
–, – stanu 462, 464, 466–469
–, rejestry 462, 464, 466–469, 667
–, sygnały 649
sterująca jednostka 31, 34
– –, implementacja układowa 658–661
– –, mikroarchitektura 667
– –, mikrooperacje 639–645
– –, model 647
– –, organizacja 681
– – procesora 638
– –, układy logiczne 659
stos, adresowanie 434
–, opis 416
–, podstawa 418
–, ramka 397
–, wskaźnik 398, 418, 465
stronicowanie 297, 302
struktury dwukońcowe 370, 421–425
suma iloczynów (SOP) 771–772
sumatory 792–793
superkomputer Cray 750
–, obliczenia wektorowe 744
superskalarne komputery 560–566, 576–584
SWP 520
sygnały danych 654
– odczytu 185
– przerwania 654
– zapisu 185
– zewnętrzne 654
sylaby 604
symboliczny program 409–410
synchroniczne liczniki 801–803
syndrom, słowo 179–180
system, magistrala 94, 103, 648
– operacyjny, definicja 272
– –, projektowanie 735–736
– –, rodzaje 275–285
– –, szeregowanie 285
– pamięci, przegląd 122–129
szesnastkowa notacja 79, 812–813

Ś

średniookresowe szeregowanie 285–286

T

tablica prawdy 767, 771
– stron 308–313
tablice Quine'a-McKluskeya 778–782
Texas Instruments 8800 692–703
Texas Instruments 8018 698, 700
Texas Instruments 8832 698–703
TLB 301–303, 309
transakcyjna warstwa FireWire 258
tranzystor 48–49
tryb adresowania 427–458
trzecia generacja komputerów 49–56

U

ujemne przepełnienie 347
ujemny niedomiar 347
układowa jednostka sterująca 671
układy kombinacyjne 771–772
– scalone 49–57
– sekwencyjne, liczniki 800–802
– –, przerzutnik D 795
– –, przerzutnik S-R 795
– –, rejestry przesuwowe 800
– –, – równoległe 798
ulotna pamięć 128
UMA, definicja 740
UNIVAC I 45
upakowane BCD 380
– mikrorozkazy 681, 684
uzupełnienie do dwóch, algorytm Bootha 340
– –, arytmetyka 325
– –, dzielenie 364
– –, mnożenie 337–342
– –, reprezentacja 324–326
– –, tablica wartości 326

V

VAX 450–452, 514–515
VLSI 528–530
von Neumanna architektura 75

W

wejście-wyjście izolowane 238
–, kanały 235, 253–255
–, magistrala 253
–, moduły 232–235
–, procesor 235, 254
–, rejestr adresu 76–77
–, – buforowy 77
–, sterownik 235
wektorowa architektura 756–758
wektorowe obliczanie 744–758
– procesory 750
– przerwanie 244
wewnętrzna magistrala procesora 651
wieloportowa pamięć 718
wielopoziomowe pamięci podręczne 145–146
wieloprocesor 714–715
– symetryczny, adresowanie 716
– –, arbitraż 716
– –, organizacja 715–718
– –, pamięć podręczna 717, 817
– –, skalowanie 715
– –, własności 713–715
– –, zarządzanie pamięcią 720
wieloprocesorowa organizacja 512, 711–713
wieloprogramowanie 281–282, 714
wielopunktowa konfiguracja 257
wielostrefowa rejestracja 195
wielowejściowe procedury 395
wielozadaniowość 281
wierzchołek stosu 394
Wilkesa sterowanie 669–671
wirtualna pamięć 128, 157, 298–305
wirtualny adres 306
wrostkowa notacja 419
wsadowe przetwarzanie 278–281
wskaźnik bieżącego okna (CWP) 520, 547
wykładnik liczb zmiennopozycyjnych 346, 353
wymiana 292–293
wyrażenia boolowskie, upraszczanie 773–782
wysokiego poziomu języki 514, 526
wywoływanie, procedura 394–397
–, rejestry rozkazów 594
–, zagnieżdżanie procedur 394
względne adresowanie 433

X

XOR 387

Z

zagnieżdżone procedury 395
zagnieżdżonych przerwań przetwarzanie 90
zamiana, algorytmy 141
zaokrąglanie 357–359
zapis danych 463
– jednoczesny 724
– opóźniony 724
–, polityka 144–145
–, sygnały 185
–, zależność 570–571
zasobów konflikt 566
zdenormalizowane liczby 359
zegar 96, 101, 119, 647
zewnętrzna pamięć podręczna 146
zmiennopozycyjna arytmetyka 322, 351–362
– mantysa 346
– normalizacja 355
– podstawa 346
– reprezentacja 322, 345–351
zmiennopozycyjne dodawanie i odejmowanie 354
– dzielenie 356
– mnożenie 355
zmiennopozycyjny standard IEEE 359
– wykładnik 346

Akronimy

ACM	Towarzystwo Techniki Komputerowej
ALU	Jednostka arytmetyczno-logiczna
ASCII	Znormalizowany amerykański kod wymiany informacji
ANSI	Amerykański Narodowy Instytut Normalizacji
BCD	Liczba dziesiętna kodowana binarnie
CD	Płyta kompaktowa
CD-ROM	Pamięć stała na płycie kompaktowej
CPU	Jednostka centralna (procesor)
CISC	Komputer o złożonej liście rozkazów
DRAM	Dynamiczna pamięć o dostępie swobodnym
DMA	Bezpośredni dostęp do pamięci
DVD	Uniwersalny dysk wideo
EPIC	Jawne równoległe przetwarzanie rozkazów
EPROM	Wymazywalna, programowalna pamięć stała
EEPROM	Elektrycznie wymazywalna, programowalna pamięć stała
HLL	Język wysokiego poziomu
I/O	Wejście-wyjście
IAR	Rejestr adresu rozkazu
IC	Układ scalony
IEEE	Instytut Elektrotechniki i Elektroniki*
ILP	Paralelizm na poziomie rozkazów
IR	Rejestr rozkazów
LRU	Najmniej ostatnio używany
LSI	Duży stopień scalenia

*Amerykańska instytucja normalizacyjna (przyp. red.)

MAR	Rejestr adresowy pamięci
MBR	Rejestr buforowy pamięci
MESI	Zmodyfikowany-wyłączny-wspólny-nieważny
MMU	Jednostka zarządzania pamięcią
MSI	Średni stopień scalenia
NUMA	Niejednorodny dostęp do pamięci
OS	System operacyjny
PC	Licznik rozkazów
PCI	System połączeń urządzeń peryferyjnych
PROM	Programowalna pamięć stała
PSW	Słowo stanu procesora
PCB	Blok sterowania procesem
RAID	Redundantna tablica niezależnych dysków
RALU	Rejestrowa jednostka arytmetyczno-logiczna
RAM	Pamięć o dostępie swobodnym
RISC	Komputer o zredukowanej liście rozkazów
ROM	Pamięć stała
SCSI	Interfejs małych systemów komputerowych
SMP	Wieloprocesor symetryczny
SRAM	Statyczna pamięć o dostępie swobodnym
SSI	Mały stopień scalenia
VLSI	Bardzo duży stopień scalenia
VLIW	Bardzo długie słowo rozkazu

Oto nowe wydanie - znacznie zmienione i rozszerzone - znakomitego podręcznika o budowie i działaniu nowoczesnych systemów komputerowych.

William Stallings, specjalista w dziedzinie sieci komputerowych i architektury komputerów, kładzie w nim nacisk na projektowanie systemu pod kątem uzyskania największej wydajności. Omawia:

- historię komputerów;
- budowę systemu komputerowego, w tym strukturę wewnętrznych połączeń (magistrale), pamięci: podręczną, wewnętrzną i zewnętrzną (także optyczną), urządzenia wejścia-wyjścia;
- procesory, z uwzględnieniem arytmetyki komputerowej, listy rozkazów, struktury i organizacji rejestrów, przetwarzania potokowego, architektur: RISC, CISC, superskalarnej i IA-64 (Itanium);
- jednostkę sterującą i sterowanie mikroprogramowe;
- przetwarzanie z wykorzystaniem wieloprocesorów i procesorów równoległych.

Wartość dzieła podnoszą doskonale dobrane przykłady najważniejszych rozwiązań.

Rozwój techniki komputerowej ma dziś podstawowe znaczenie dla cywilizacji. Wydawnictwa Naukowo-Techniczne gorąco polecają tę książkę studentom elektroniki i informatyki, bo to oni będą w przyszłości wytyczać kierunki tego rozwoju.

Cena 96,- zł
ISBN 83-204-2892-0

